Masking Curves

Side-Channel Attacks on Elliptic Curve Cryptography and Countermeasures

Louiza Papachristodoulou

ISBN: 978-94-93118-05-8
NUR: 980

Typeset using LaTeX
Cover design by Wendy Schoneveld (http://www.wenzid.nl/)
Printed by ProefschriftMaken || (www.proefschriftmaken.nl/)

Radboud University

The work in this thesis is funded by Technology Foundation STW through project 12624-SIDES. The author was employed by the Radboud University, Nijmegen.

Masking Curves
Side-Channel Attacks on Elliptic Curve Cryptography and Countermeasures

DOCTORAL THESIS

to obtain the degree of doctor
from Radboud University Nijmegen
on the authority of the Rector Magnificus, prof. dr. J.H.J.M. van Krieken,
according to the decision of the Council of Deans
to be defended in public on **Friday, March 22, 2019**
at **10.30 hours**.

by

Louiza Papachristodoulou

Born on July 19, 1984
in Marousi, Greece

Supervisor:

Prof. dr. Lejla Batina

Doctoral Thesis Committee:

Prof. dr. ir. Joan Daemen	
Prof. dr. ir. Bart Preneel	Katholieke Universiteit Leuven, Belgium
Dr. ir. Marc Joye	One Span, Belgium
Prof. dr. Elisabeth Oswald	University of Bristol, United Kingdom
Prof. dr. Kazuo Sakiyama	University of Electro-Communications, Japan

Masking Curves
Side-Channel Attacks on Elliptic Curve Cryptography and Countermeasures

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. J.H.J.M. van Krieken,
volgens besluit van het college van decanen
in het openbaar te verdedigen op vrijdag 22 maart 2019
om **10.30 uur** precies.

door

Louiza Papachristodoulou

geboren op 19 juli 1984
te Marousi, Griekenland

Promotor:

Prof. dr. Lejla Batina

Manuscriptcommissie:

Prof. dr. ir. Joan Daemen	
Prof. dr. ir. Bart Preneel	Katholieke Universiteit Leuven, België
Dr. ir. Marc Joye	One Span, België
Prof. dr. Elisabeth Oswald	University of Bristol, Verenigd Koninkrijk
Prof. dr. Kazuo Sakiyama	University of Electro-Communications, Japan

To my family &
In memory of my father
Panagiotis Papachristodoulou

Acknowledgements

An amazing journey is coming to an end after almost 5 years. It has been an incredible experience with a lot of interesting people, collaborations, workshops, conferences, an emotional roller-coaster and above all a fulfiling journey in the world of cryptography. A constant alternation between social interactions and long, quiet hours in the lab. I am grateful for everyone that I met in the most exciting path of my life so far, and at this point I would like to thank the people who were by my side and made everything possible.

First of all, I would like to thank my partner Christos Bousis for brightening the most dark years of my life, for being my "seal" mate and for believing in me. Thank you for helping with the figures of this thesis, for your patience and your unconditional love!

Lejla, my promotor, supervisor, ex-landloard and most importantly my friend thank you for believing in me more than I was believing in myself and for all the opportunities you gave me to develop myself and expand my knowledge. Thank you for giving me the chance to fulfil my dream of doing a PhD, for the nice working environment and for all the unforgetable moments during our trips. You are the best supervisor I could have ever asked for!

During the past 5 years I was priviledged to work with some of the best researchers and professors in our field. I would like to thank my coauthors Dan Bernstein, Łukasz Chmielewski, Jean-Christophe Courrège, Jean-Luc Danger, Margaux Dugardin, Apostolos Fournaris, Sylvain Guilley, Daira Hopwood, Andreas Hülsing, Tanja Lange, Nele Mentens, Zakaria Najm, Ruben Niederhagen, Elif Özgen, Kostas Papagiannopoulos, Michael Schneider, Peter Schwabe, Nikolaos Sklavos, Carine Therond, Michael Tunstall and Zooko Wilcox-O'Hearn for sharing their knowledge with me and for working hard for our publications. Special thanks to Andy for being also a good friend and neighbor in Eindhoven, to Peter for always being excited to help with his amazing coding skills in exchange with a good beer, to Łukasz for his effort and committment in our first paper and for his support ever after - many things would have been impossible without his contribution, to Apostolos for the nice times in Patras and in conferences, the long Skype calls while collaborating remotely and his eagerness to help even in the weekends, and to Mike for being a great colleague at Rambus and willing to work together even when the distance and

time difference made our research results hard to share. I am also grateful to my committee for reading and accepting my thesis; in particular, to Joan Daemen for his help and guidance, to Elisabeth Oswald for her critical remarks, to Bart Preneel and Marc Joye for their detailed comments that really improved my manuscript and to Kazuo Sakiyama for his comments and willingness to attend my defense all the way from Japan.

I am grateful for all the research visits that I did to various groups around the world and for having the change to work with the best researchers in cryptography. I would like to thank Jean-Luc Danger for hosting me at Télécom ParisTech twice and for his continuous support after that. Pankaj Rohatgi, it was an honor to have as manager a person that I already admired from the papers I read when starting my PhD; thank you for giving me the chance to work for 3 months at Rambus in the DPA group. Sami Saab thank you for your supervision while at Rambus and for teaching me the interesting parts of signal processing. Nikolaos Sklavos thank you for hosting me in the University of Patras and for all the fun times during TRUDEVICE workshops. The network of TRUDEVICE working group gave me the opportunity to attend a lot of interesting workshops and spend some time doing research abroad, therefore, I would like to thank the organisers and committee members for that. Special thanks to Francesco Regazzoni and Ilia Polian, not only for the interesting and fun times at the workshops, but also for giving me the inspiration for my thesis cover page. Aggelos Kiayias thank you for hosting me in the University of Athens and in the University of Edinburgh for 2 months. I am grateful to you and your colleagues, Yiannis Tselekounis, Giorgos Panagiotakos, Katerina Samari, Thomas Zacharias, Myrto Arapinis and Adrianna Gkaniatsou; thank you guys for supporting me and being my friends for those 2 months and afterwards, it meant a lot to me. Finally, I would like to thank Yuval Yarom for hosting me in the University of Adelaide, for his support, the interesting collaboration and for making this first experience Down-Under unforgetable.

At the beginning of my PhD, I was also working at Compumatica Secure Networks. I would like to thank my former colleagues Cees, Peter, Jan, Klaus, Sander, Rob, Eduardo, my manager Ries van Son and our director Petra van Schayik for supporting me in this decision and for making it possible to combine both jobs for one year. Cees Jansen, an amazing person that I am priviledged to know for the past 8 years, gave me the inspiration and motivation to start this journey. Thank you Cees for all the interesting discussions at Compumatica, for connecting me to Lejla, for introducing me to cryptographic and mystic worlds and for your support throughout all these years.

I was super lucky to work in the Digital Security Group at Radboud and to have amazing colleagues. Bart Jacobs and Erik Poll thank you for accepting me in the group and for all your support. Kostas, thank you for being a real friend all these years, for being the funniest office-mate, for hosting me in Nijmegen with Maria, for all your help in the lab, and all the nice moments we shared in our common journey,

including the unforgetable one in India! Joost Renes thank you for being my friend, for all your support, for correcting my chapter on ECC, for helping me with the samenvatting, for your patience to speak Dutch with me and for the zombies in Crotia! I am very happy and proud to have you as my paranymph! I would also like to thank all the members of the cesca-lab, Pedro, Niels, Ko, Joost, Stjepan, Baris, Veelasha and my office-mates Anna and Simona for the nice moments in the lab and in the conferences, especially on the pirate-boat! Gergely, I am grateful to know you for 7 years, thank you for all your advice, guidance, nice trips and continuous support as a colleague and as a friend who really cares. My favorite two Australians, Ben Smith and Craig Costello, might not be officially members of the DS group, but they are often honoured guests! Thank you both for all your help and guidance from the beginning of my PhD until now!

On a more personal note, I am grateful for the amazing housemates that supported me in our daily life and for being by my side as sisters. Thank you Lena, Kyveli, Eleni, Ellidora and Pelagia for everything, I was really lucky to come back to our gezellig home after my trips and have some friends-like-family to talk to. Lena, thank you also for accepting to be my paranymph and for being always so proud of me! I am lucky to have many friends in Eindhoven who really supported me before and during my PhD, Maria, Chrysostomos, Katerina, Giorgos, Dimitra, Petros, Chrysoula, Nikos Sotiropoulos, Nikos Mpatalas, Agis, Dwra, Andrea, Chiara, Döndü, Seray, Athena, Andreea, Thimios, Costas, thank you all guys for making Eindhoven such a nice place to live! I would really like to thank my friends from Greece. Giorgo Skoumako I am grateful for having you always by my side, in fun and difficult times, supportive and patient, believing in me since high-school; happy that we are both doctors now! Glykeria, Despoina, Semele thank you for our fun times at the university, for our holidays and for everything that we shared all these years as a family. I would also like to thank my schoolmates Chrysoula, Litsa, Natasa, Natalia, Chara, Fani and koumparous Eleni, Tasos, Alkistis, Andreas for being my friends for more than 20 years; I am lucky to still have you all in my life, even if we do not live in the same country.

Last but not least, I would like to thank my family, these amazing people who were always supportive and caring, my dad Panagiotis, my mum Vicky and my brother Ian. Nothing would be possible without them. Above all, I would like to thank the only person who will be physically absent from my defense, but he would be the proudest person in the world to be there, my dad, for everything that he offered to me and for making me, together with mum, the person that I am today. Σας ευχαριστώ πολύ για ό,τι έχετε κάνει για μένα, συγνώμη που είμαι μακριά, αλλά ελπίζω να σας κάνω περήφανους. Σας αγαπώ πολύ!

Louiza Papachristodoulou
Athens, January 2019

Abstract

In a world where cryptographic devices take part in our everyday activities, it becomes very important to secure these devices against malicious users. To achieve this goal, security research is devoted to the design of secure systems by exploring new attack techniques ahead of adversaries and proposing efficient countermeasures against those attacks.

In this thesis, we focus on the security of public key cryptographic algorithms, and more precisely on Elliptic Curve Cryptography (ECC). ECC is broadly used for implementing asymmetric cryptographic protocols in embedded devices due to the small key length, low memory and power requirements compared to equivalent RSA implementations.

Physical attacks constitute a common threat against the security of embedded devices. An adversary can recover secret information by measuring the execution time of an algorithm, its power consumption, the electromagnetic (EM) emanations, the photon emissions or other physical quantities that could give a meaningful physical leakage. These attacks, which exploit the physical leakage of secret information from cryptographic devices, are characterized as *side-channel analysis* (SCA) attacks. Within this area of attacks, there are various methods of analysis, such as Simple Power Analysis (SPA), Differential Power Analysis (DPA) and Collision Analysis (CA). SPA uses a single power trace or a few traces and draws conclusions over the secret data by this instantaneous power consumption of a single run of the algorithm under attack. DPA uses statistical methods to extract information from multiple traces over different time periods. CA exploits the leakage of two portions of traces when the same intermediate values are used. We observe that the trend of SCA attacks is shifting more towards collision and horizontal types attacks as known randomization-based countermeasures are effective in protecting cryptographic algorithms against SPA and DPA attacks.

In this aspect, the contribution of this thesis is two-fold. We first present a new, powerful attack technique, called Online Template Attacks (OTA), with broad applicability in various ECC implementations. OTA leans towards the horizontal type of attacks, since it needs only one trace from the device under attack, in order to retrieve the secret key. Therefore, common countermeasures, such as randomization of the scalar, are not effective against this attack. We show practical results on vari-

ous platforms and elliptic curves and at the same time, we present countermeasures that could prevent OTA.

In the second part of this thesis, we present and evaluate algorithmic and numerical countermeasures, in order to give new insights in secure implementations of ECC. More precisely, we first present the Boolean-XOR Splitting technique, a countermeasure inspired by masking techniques that are common in symmetric algorithms. Boolean-XOR Splitting can be applied to exponentiation and scalar multiplication algorithms with minimal impact on performance, since it is based on Boolean shares. It is shown how an exponent can be efficiently split into two shares, where the exponent is the XOR sum of the two shares, typically requiring only an extra register and a few register copies per bit. Our novel exponentiation and scalar multiplication algorithms can be randomized for every execution and combined with other blinding techniques, maintaining at the same time the regularity feature. In this way, both the exponent and the intermediate values can be protected against various types of side-channel attacks. We perform a security evaluation of our algorithms using the Mutual Information framework and we verify formally that they are secure against first-order attacks. The resistance of the proposed algorithms against side-channel attacks is practically verified with Test Vector Leakage Assessment (TVLA).

The Residue Number System (RNS) arithmetic is gaining grounds in public key cryptography, because it offers fast, efficient and secure implementations over large prime fields or rings of integers. In cryptographic applications of public key cryptography, the recommended bit length varies between 256 bits (for ECC) to 3072 bits (for RSA). RNS-based implementations offer the possibility to perform the required modulo operations on smaller numbers by distributing the integer operations on the residue values. These computations can, furthermore, be executed independently over each modulus, allowing parallelization of the calculations. We explore the potentials of this numerical representation as a countermeasure for secure ECC implementations. More specifically, variations of RNS-based Montgomery Power Ladder scalar multiplication algorithms are evaluated using a generic and thorough methodology based on the TVLA and template attacks. Our data and location dependent template attacks show that even protected implementations of RNS are vulnerable to these attacks.

Samenvatting

In een wereld waarin cryptografische apparaten deelnemen aan onze dagelijkse activiteiten, wordt het erg belangrijk om deze apparaten te beveiligen tegen kwaadwillende gebruikers. Om dit doel te bereiken, wordt veiligheidsonderzoek gewijd aan het ontwerpen van veilige systemen door nieuwe aanvalstechnieken te verkennen voorafgaand aan vijandige gebruikers en efficiënte tegenmaatregelen tegen die aanvallen voor te stellen.

In dit proefschrift richten we ons op de veiligheid van cryptografische algoritmen met openbare sleutels, en specifiek op Elliptic Curve Cryptografie (ECC). ECC wordt algemeen voor het implementeren van asymmetrische cryptografische protocollen in ge"integreerde systemen gebruikt vanwege de kleine sleutellengte, lage geheugen- en stroomvereisten vergeleken met gelijkwaardige RSA-implementaties.

Fysieke aanvallen vormen een veelvoorkomende bedreiging voor de beveiliging van ingebedde apparaten. Een kwaadwillende gebruiker kan geheime informatie herleiden door de uitvoeringstijd van een algoritme, het stroomverbruik, de elektromagnetische (EM) emanaties, de fotonenemissies of andere fysieke grootheden te meten die een zinvolle fysieke lekkage zouden kunnen geven. Deze aanvallen, die gebruikmaken van de fysieke lekkage van geheime informatie van cryptografische apparaten, worden gekenmerkt als *side-channel analysis* (SCA) aanvallen. Binnen dit gebied van aanvallen zijn er verschillende analysemethoden, zoals Simple Power Analysis (SPA), Differential Power Analysis (DPA) en Collision Analysis (CA). SPA maakt gebruik van een of enkele stroomsporen en trekt conclusies over de geheime gegevens door dit directe energieverbruik van een enkele uitvoering van het aangevallen algoritme. DPA gebruikt statistische methoden om informatie uit meerdere sporen te extraheren over verschillende tijdsperioden. CA exploiteert het lekken van twee delen van sporen wanneer dezelfde tussenliggende waarden worden gebruikt. We zien dat de trend van SCA-aanvallen meer verschuift naar botsingen en horizontale aanvallen, omdat bekende tegenmaatregelen op basis van randomisatie effectief zijn in het beschermen van cryptografische algoritmen tegen SPA- en DPA-aanvallen.

In dit aspect is de bijdrage van dit proefschrift tweeledig. We presenteren eerst een nieuwe, krachtige aanvalsmethode, genaamd Online Template Attacks (OTA), met brede toepasbaarheid in verschillende ECC-implementaties. OTA leunt in de

richting van het type horizontale aanvallen, omdat het slechts één spoor van het aangevallen apparaat nodig heeft om de geheime sleutel te herleiden. Daarom zijn veelvoorkomende tegenmaatregelen, zoals randomisatie van de scalair, niet effectief tegen deze aanval. We tonen praktische resultaten op verschillende system en elliptische krommen en tegelijkertijd presenteren we tegenmaatregelen die OTA kunnen voorkomen.

In het tweede deel van dit proefschrift presenteren en evalueren we algoritmische en numerieke tegenmaatregelen om nieuwe inzichten te geven in veilige implementaties van ECC. Om precies te zijn, presenteren we eerst de Boolean-XOR Splitting techniek, een tegenmaatregel die is gebaseerd op masking, die veel in symmetrische algoritmen gebruikt wordt. Boolean-XOR Splitting kan worden toegepast op exponentiatie en scalaire vermenigvuldigingsalgoritmen met minimale impact op de prestaties, omdat deze is gebaseerd op Booleaanse shares. Er wordt getoond hoe een exponent efficiënt kan worden gesplitst in twee delen, waarbij de exponent de XOR-som van de twee delen is, waarbij doorgaans slechts een extra register en enkele registerkopieën per bit nodig zijn. Onze nieuwe exponentiatie en scalaire vermenigvuldigingsalgoritmen kunnen voor elke uitvoering worden gerandomiseerd en gecombineerd met andere blinderingstechnieken, waarbij tegelijkertijd de regelmatigheidsfunctie behouden blijft. Op deze manier kunnen zowel de exponent als de tussenliggende waarden worden beschermd tegen verschillende soorten side-channel aanvallen. We voeren een veiligheidsevaluatie van onze algoritmen uit met behulp van het Mutual Information-framework en we verifiëren formeel dat ze beveiligd zijn tegen aanvallen van de eerste orde. De weerstand van de voorgestelde algoritmen tegen side-channel aanvallen is praktisch geverifieerd met Test Vector Leakage Assessment (TVLA).

De Residue Number System (RNS) wint terrein in cryptografie met openbare sleutels, omdat dit rekenkunde snelle, efficiënte en veilige implementaties biedt over grote priemlichamen of ringen van gehele getallen. Bij cryptografische toepassingen van cryptografie met openbare sleutels varieert de aanbevolen bitlengte tussen 256 bits (voor ECC) tot 3072 bits (voor RSA). Op RNS gebaseerde implementaties bieden de mogelijkheid om de vereiste modulus-bewerkingen in kleinere aantallen uit te voeren door de gehele bewerking over de residuwaarden te verdelen. Deze berekeningen kunnen bovendien voor elke modulus onafhankelijk worden uitgevoerd, waardoor parallellisatie van de berekeningen mogelijk is. We verkennen de mogelijkheden van deze numerieke weergave als een tegenmaatregel voor veilige ECC-implementaties. In het bijzonder, variaties van RNS-gebaseerde Montgomery Power Ladder scalaire vermenigvuldigingsalgoritmen worden geëvalueerd met behulp van een generieke en grondige methodologie op basis van de TVLA- en sjabloonaanvallen. Over de beveiliging van RNS tonen onze gegevens en locatie afhankelijke sjabloonaanvallen aan dat zelfs een beschermde implementatie kwetsbaar is.

Περίληψη

Η λειτουργία της κοινωνίας μας είναι πλέον άρρηκτα συνδεδεμένη με το διαδίκτυο και κατ΄ επέκταση με την εξέλιξη σε επιστήμες όπως πληροφορική, πρωτόκολλα επικοινωνίας και ασφάλεια πληροφοριών. Σε έναν κόσμο όπου οι συσκευές κρυπτογραφίας αποτελούν τμήμα της καθημερινότητας μας, είναι πολύ σημαντικό να προστατεύσουμε αυτές τις συσκευές από κακόβουλους χρήστες. Η έρευνα ως προς την ασφάλεια αυτών των συσκευών και τον ευρύτερο σχεδιασμό συστημάτων ασφαλείας επικεντρώνεται στην 'εξερεύνηση' νέων τρόπων επιθέσεων, προλαμβάνοντας τις επιθέσεις από κακόβουλους χρήστες (χάκερς), και στην πρόταση αντιμέτρων που μπορούν να προστατεύσουν τα συστήματα από αυτές τις επιθέσεις.

Σε αυτήν την διπλωματική διατριβή, επικεντρωνόμαστε στην ασφάλεια κρυπτογραφικών αλγορίθμων δημόσιου κλειδιού, και ειδικότερα στην Κρυπτογραφία Ελλειπτικών Καμπυλών (ECC). Η Κρυπτογραφία Ελλειπτικών Καμπυλών χρησιμοποιείται ευρέως για την υλοποίηση πρωτοκόλλων ασύμμετρης κρυπτογραφίας σε ενσωματωμένες συσκευές λόγω του μικρού μήκους κλειδιού και των χαμηλών απαιτήσεων σε μνήμη και ενέργεια συγκριτικά με ισοδύναμες υλοποιήσεις του αλγορίθμου RSA.

Οι φυσικές επιθέσεις αποτελούν μία συνηθισμένη απειλή για την ασφάλεια ενσωματωμένων συσκευών. Κάποιος που σκοπεύει να επιτεθεί σε μία συσκευή μπορεί να ανακαλύψει μυστικές πληροφορίες μετρώντας τον χρόνο εκτέλεσης ενός αλγορίθμου, την κατανάλωση ρεύματος, την ηλεκτρομαγνητική ακτινοβολία και άλλες φυσικές ιδιότητες του κυκλώματος, οι οποίες μπορούν να δώσουν χρήσιμες και αξιοποιήσιμες πηγές 'διαρροής' πληροφορίας. Αυτού του τύπου οι επιθέσεις, που εκμεταλλεύονται τη φυσική διαρροή μυστικών πληροφοριών από κρυπτογραφικές συσκευές, χαρακτηρίζονται ως επιθέσεις παράπλευρου καναλιού (side-channel attacks-SCA). Στα πλαίσια αυτού του είδους επιθέσεων, υπάρχουν διάφορες μέθοδοι ανάλυσης των δεδομένων, όπως για παράδειγμα η Απλή Ανάλυση Ρεύματος (SPA), Διαφορική Ανάλυση Ρεύματος (DPA) και Ανάλυση Συγκρούσεων (CA). Η ανάλυση SPA χρειάζεται ένα ή ελάχιστα ηλεκτρικά σήματα ώστε να βγάλει συμπεράσματα για τα μυστικά δεδομένα από τη στιγμιαία κατανάλωση ρεύματος κατά τη διάρκεια μίας εκτέλεσης του αλγορίθμου υπό επίθεση. Η ανάλυση DPA χρησιμοποιεί στατιστικές μεθόδους για να εξάγει πληροφορίες από

πολλαπλά σήματα σε διαφορετικές χρονικές περιόδους. Η ανάλυση CA αξιοποιεί τη διαρροή πληροφοριών από δύο τμήματα του σήματος όταν χρησιμοποιούνται οι ίδιες ενδιάμεσες τιμές. Τα τελευταία χρόνια, παρατηρούμε ότι επικρατεί η τάση να χρησιμοποιούνται επιθέσεις παράπλευρου καναλιού οριζόντιου τύπου (SPA, CA), καθώς γνώστα αντιμέτρα που βασίζονται σε τυχαιοποίηση μεταβλητών είναι αποτελεσματικά για την προστασία αλγορίθμων ενάντια σε SPA και DPA επιθέσεις.

Από αυτήν την σκοπιά, η συνεισφορά αυτής της διπλωματικής είναι διττή. Καταρχάς, παρουσιάζουμε μία νέα και ισχυρή επίθεση, την επονομαζόμενη Online Template Attack (OTA), που μπορεί να χρησιμοποιηθεί ευρέως σε διάφορους αλγορίθμους ελλειπτικών καμπυλών. Η OTA κλίνει προς τις επιθέσεις οριζόντιου τύπου, αφού χρειάζεται μόνο ένα σήμα από την συσκευή υπό επίθεση, ώστε να ανακτήσει το μυστικό κλειδί. Συνεπώς, συνήθη αντιμέτρα, όπως η τυχαιοποίηση του πολλαπλασιαστή (scalar) ή του εκθέτη (exponent), δεν είναι αποτελεσματικά ενάντια σε αυτήν την επίθεση. Παρουσιάζουμε την εφαρμογή της επίθεσης αυτής σε διαφορετικές πλατφόρμες και ελλειπτικές καμπύλες, ενώ ταυτόχρονα προτείνουμε αντιμέτρα που θα μπορούσαν να χρησιμοποιηθούν για να προστατεύσουν τις συσκευές από επιθέσεις τύπου OTA.

Στο δεύτερο μέρος αυτής της διατριβής παρουσιάζουμε και αξιολογούμε αλγοριθμικά και αριθμητικά αντιμέτρα, με σκοπό να δώσουμε νέες ιδέες για την ασφαλή υλοποίηση αλγορίθμων ελλειπτικών καμπυλών. Πιο συγκεκριμένα, παρουσιάζουμε πρώτα την τεχνική Boolean-XOR Splitting, ένα αντιμέτρο εμπνευσμένο από τις τεχνικές masking, που χρησιμοποιούνται ευρέως σε αλγορίθμους συμμετρικής κρυπτογραφίας. Η τεχνική Boolean-XOR Splitting μπορεί να χρησιμοποιηθεί σε εκθετικούς και πολλαπλασιαστικούς αλγορίθμους με ελάχιστες επιπτώσεις στην απόδοση και ταχύτητα υλοποίησης, επειδή βασίζεται σε Boolean πράξεις. Δείχνουμε πώς ένας εκθέτης μπορεί να χωριστεί αποτελεσματικά σε δύο μέρη, με τέτοιον τρόπο ώστε ο εκθέτης να είναι το XOR-άθροισμα των δύο μερών. Αυτό απαιτεί συνήθως μόνο έναν επιπλέον καταχωρητή και μερικές αντιγραφές καταχωρητών για κάθε bit. Η νέα και πρωτότυπη μέθοδος που προτείνουμε μπορεί να εφαρμοστεί και σε εκθετικούς (RSA) και σε πολλαπλασιαστικούς (ECC) αλγορίθμους, μπορεί να δώσει τυχαία αποτελέσματα για κάθε εκτέλεση του αλγορίθμου, καθώς και να συνδυαστεί με τεχνικές blinding, διατηρώντας παράλληλα την ιδιότητα της ομαλότητας και κανονικότητας του αλγορίθμου. Με αυτόν τον τρόπο, τόσο ο εκθέτης (ή πολλαπλασιαστής) όσο και οι ενδιάμεσες τιμές προστατεύονται ενάντια σε διάφορους τύπους SCA. Παρουσιάζουμε μια αξιολόγηση των αλγορίθμων μας ως προς την ασφάλεια βασιζόμενοι στο πλαίσιο του Mutual Information και αποδεικνύουμε ότι οι αλγόριθμοι είναι ασφαλείς σε επιθέσεις πρώτου τύπου. Η αντίσταση των αλγορίθμων σε επιθέσεις παράπλευρου καναλιού επιβεβαιώνονται και με την εφαρμογή των στατιστικών τεστ Test Vector Leakage Assessment (TVLA).

Το σύστημα αριθμητικής υπολοίπων (Residue Number System-RNS) χρησι-

μοποιείται ευρέως στην κρυπτογραφία δημόσιου κλειδιού, επειδή προσφέρει γρήγορες, αποτελεσματικές και ασφαλείς υλοποιήσεις σε πεδία μεγάλων πρώτων αριθμών και δακτύλιους ακεραίων. Στις εφαρμογές κρυπτογραφίας δημόσιου κλειδιού, τα προτεινόμενα μήκη κλειδιών κυμαίνονται μεταξύ 256 bits για το ECC και 3072 bits για το RSA. Οι υλοποιήσεις που βασίζονται στο RNS δίνουν τη δυνατότητα να εκτελούνται οι απαραίτητες πράξεις με υπόλοιπα σε μικρότερους αριθμούς, διανέμοντας τις πράξεις ακεραίων στις αντίστοιχες τιμές τους ως υπόλοιπα. Αυτοί οι υπολογισμοί μπορούν επίσης να εκτελεστούν ανεξάρτητα για κάθε υπόλοιπο, επιτρέποντας την παράλληλη εκτέλεση των υπολογισμών. Ερευνούμε τις δυνατότητες αυτής της αριθμητικής αναπαράστασης ως αντιμέτρο για υλοποιήσεις ECC. Πιο συγκεκριμένα, αξιολογούμε παραλλαγές υλοποιήσεων του πολλαπλασιασμού Montgomery Power Ladder βασιζόμενοι σε μια γενική και διεξοδική μεθοδολογία, χρησιμοποιώντας TVLA και επιθέσεις τύπου templates. Όσον αφορά την ασφάλεια του RNS οι template επιθέσεις μας στα δεδομένα και στην θέση του μυστικού κλειδιού, δείχνουν ότι ακόμα και προστατευμένες υλοποιήσεις είναι ευάλωτες σε τέτοιου τύπου επιθέσεις.

Contents

List of Abbreviations

ADD	Addition (on elliptic curves)
CA	Collision Analysis
CPA	Correlation Power Analysis
CRT	Chinese Remainder Theorem
DPA	Differential Power Analysis
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
EdDSA	Edwards Digital Signature Algorithm
EM	ElectroMagnetic
DBL	Doubling (on elliptic curves)
FIPS	Federal Information Processing Standards
FWER	Family-Wise Error Rate
HD	Hamming Distance
HW	Hamming Weight
IPsec	Internet Protocol Security
IT	Information Technology
k-NN	k-Nearest Neighbor
MI	Mutual Information
ML	Machine Learning
MPL	Montgomery Power Ladder
MSB	Most Significant Bit

MSW	Most Significant Word
NAF	NonAdjacent Form
NIST	National Institute of Standards and Technology
OTA	Online Template Attacks
RDA	Refined Doubling Attack
RNS	Residue Number System
RSA	Rivest-Shamir-Adleman
SCA	Side-Channel Analysis
SEMA	Simple ElectroMagnetic Attacks
SHA	Secure Hash Algorithms
SPA	Simple Power Analysis
SSH	Secure SHell
SSL	Secure Socket Layer
SVM	Support Vector Machine
TA	Template Attacks
TLS	Transport Layer Security
TVLA	Test Vector Leakage Assessment

List of Figures

Chapter 1

Introduction

Η αρχή είναι το ήμισυ του παντός - Well begun is half done

Aristotle

Cryptography is characterized as the art of writing secret codes. In this regard, cryptanalysis is the art of breaking them.

Cryptography, by its definition, is associated with secrets. It comes from the Greek words κρυπτό (krypto, hidden) and γράφειν (grafein, to write). From the ancient ways of encrypting messages with Scytale [Kel98] to the modern ways based on computationally hard to solve problems, the aim of encryption is the same: the sender and the receiver want to communicate securely over an insecure channel and keep their communication secret from adversaries. Cryptographic research is advancing over the last 40 years, in order to provide secure means of communication over the most popular and insecure channel, i.e. the Internet.

Cryptanalysis, the study on ciphertexts for retrieving the secret messages without knowing the key, is an active research area that offers many techniques to break into security systems. The aim of cryptanalytic techniques is to find efficient ways to break cryptographic algorithms by mathematical analysis either of the algorithms or by relations between inputs (plaintexts) and outputs (ciphertexts) of the algorithms. Linear cryptanalysis takes advantage of high probability occurrences of linear expressions involving plaintext bits, while differential cryptanalysis exploits the high probability of certain occurrences of plaintext differences and differences in the output of the algorithm. They are both a type of chosen plaintext attack, in the sense that the attacker is able to select inputs and examine outputs in an attempt to derive the key. The attacker is usually assumed to have access to a "black-box", where he is able to choose what input to feed in the algorithm, but he does not have any specific information of how the algorithm works. By finding vulnerabilities in the algorithms or by weakening the security level of a protocol, researchers are capable

to propose countermeasures to protect security systems.

Side-channel analysis (SCA) can be considered as the most applied part of cryptanalysis, where the adversary takes advantage of implementation leakage produced by devices executing a cryptographic algorithm. While the goal of a side-channel attacker is the same as a cryptanalyst, i.e. to find vulnerabilities and break a secure algorithm, the means that they use to achieve this goal are different. As opposed to "black-box" and mathematical properties, SCA deals with a "grey-box" attack design and physical properties. Exploitable leakage comes from the execution time, the power consumption, the electromagnetic emanations or even by the noise produced by electric components during an execution of an algorithm. All this side-channel information, which can be obtained during execution of the algorithms, give to the adversary some extra insight about the system and that is the reason why SCA is considered as "gray-model".

Despite the advances in cryptographic research, the security breaches and attacks are an emerging threat for users and enterprises. Online fraud appears in many forms, email spam, phishing, online scams, hacking users' accounts and many others. This sort of attacks are not related to side-channel attacks, since they exploit mainly protocol vulnerabilities, resulting in buffer overflows, denial-of-service, privilege escalation etc. Debit and credit cards are common targets for these attacks with huge economic impact. According to the Dutch payments' association report of 2018,[1] the total loss from payment frauds in the Netherlands reached 12.8 million euros in 2017 [Ned18].

Side-channel information can be exploited when protocol vulnerabilities are combined with hardware equipment, in order to attack a system. Side-channel analysis is performed on microcontrollers used for embedded applications. Therefore, they affect a broad range of products, such as smart cards, mobile phones, hardware wallets, RFID tags and automotive electronic control units to name a few. An attack of this kind with high impact was the so-called "EMV PIN verification - wedge vulnerability" affecting at least 730 million payment cards in circulation [MDAB10]. The group from the University of Cambridge performed a man-in-the-middle attack, which suppresses the PIN Verify command to the card, and tells the terminal that the PIN has been verified correctly, even during online transactions. Researchers from Radboud University report that EMV vulnerabilities can be exploited in payment smart cards, affecting billion of cards worldwide [CGdR+15, vdBOYPdR]. The fact that EMV is an old protocol suite using outdated symmetric key algorithms (3DES instead of AES) and requiring backwards compatibility between the different protocol variants, make it very hard to provide fixes for secure transactions.

Regarding payment systems, side-channel attacks pose a serious threat against hardware wallets. With the remarkable cryptocurrency expansion at the end of 2017,

[1] The website gives the tables and numbers when the Dutch language is selected - switching the website to English would make those numbers disappear.

the need to store and trade cryptocurrency in a secure way emerged. During the past two years, companies producing hardware wallets experience exponential increase in demand [Int]. Hardware wallets need to be designed in a secure way offering defense in depth and security in layers and storing keys on a secure element. Hoenicke [Hoe15] presented a practical side-channel attack on a Trezor wallet, where he was able to recover the private key using the information leaked during the public key generation, when the wallet was not protected by a passphrase. Updating the firmware to the latest version is enough to defend against this attack, but still the threat of side-channel attacks against hardware wallets is not eliminated.

Pay TV is another industry that is affected by side-channel attacks. The number of pay TV subscribers worldwide is projected to pass the one billion mark by 2020, growing by 92% since 2014 [SP]. Pay TV decoders use smart cards, in order to control video access. There are various techniques or videos online that show how to hack such cards. Witteman presented [Wit17] a card sharing attack (a relay attack to avoid paying TV subscription fees) of 1 million cards, which would result in a loss of 90 million euros from the pay TV industry. However, a major shift in viewing habits towards streaming services such as Netflix and Amazon Prime appeared in 2018. When subscriptions to streaming services overtake pay TV users, protection against side-channel attacks will switch to software protection for the TV industry.

In the beginning of 2018, two critical vulnerabilities in modern processors appeared. Spectre [KGG$^+$18] and Meltdown [LSG$^+$18] are cache timing side-channel attacks that exploit hardware vulnerabilities of processors, in order to either read data or get access to secure memory locations with no access privileges. Passwords stored on personal computers, mobile devices or the cloud can leak due to these attacks. A year after the publication of the two original flaws, more exploitable weaknesses related to this class of "speculative execution" have been published, but an efficient way to fix them is still not proposed. This is due to the fact that software patches are not enough; we need to conceptually rethink how processors are made [New18].

Automotive is a fast growing industry that will be affected by SCA attacks. The Internet of Things (IoT) in automotive market is projected to grow at a compound annual growth rate (CAGR) of 27.55% leading to a global revenue of 104 billion dollars by 2023 [RM, aut]. Quite a lot of researchers, organizations and companies worldwide are working towards the establishement of a secure IoT automotive ecosystem. It is challenging to provide hardware security, cloud security, attack detection systems and privacy protection at the same time. Dedicated Hardware Security Modules (HSM) are usually deployed to provide security features in the microcontrollers used in automotive industry. However, many publications (for instance lectures from the escar conferences [Esc], the papers from Milburn et al. [MCW$^+$18, MT18] and Jain et al. [JWAG18] to name a few) showed that side-channel attacks are still possible in this field and new types of countermeasures are needed, in order to provide secure automotive solutions.

How is it possible to have this huge amount of attacks, when thousands of cryptographic experts work for the security of digital systems?

The answer lies in the implementation details. Cryptographic devices leak information and when this leakage can be exploited, then compromising the system and recovering secrets is unavoidable. The adversaries can have an easier task, when countermeasures are not applied, sensitive data are sent in clear, authentication mechanisms are not implemented correctly or protocol flaws are not fixed.

1.1 Motivation

Common daily actions would not be possible without the advances of research on cryptography. For instance online banking, ATM withdrawal, secure texting (Viber, WhatsApp, Signal) rely on fundamental cryptographic protocols, such as AES for encryption, digital signatures (DSA, ECDSA) or key exchange protocols (Diffie-Hellman). The vast majority of adult population carries a few cryptographic devices every day, such as payment cards and mobile phones. Sensitive private information is stored in cryptographic devices and the main goal of cryptographers is to protect them against various types of attacks.

The security of embedded devices, even devices with dedicated cryptographic processors, is dependent on resilience against side-channel attacks (SCA). Power consumption [KJJ99], electromagnetic emanations [GMO01, QS01, AARR03] and other forms of side-channel information leakage can expose the sensitive data of an implementation (usually a cryptographic key), through the use of various types of SCAs, such as Simple Power Analysis (SPA), Differential Power Analysis (DPA) [KJJ99] and Template attacks [CRR02, MO09]. The above attacks can be characterized as passive, in the sense that the adversary observes the leakage of physical quantities and draws conclusions about the secret data by analyzing this leakage. Fault Attacks (FA) are active attack scenarios in the area of SCA, and they usually require more sophisticated equipment, in order to successfully inject a fault. The aim of the adversary is to inject a fault in the cryptographic operation that manipulates the key; this fault can be used to alter the expected execution order of the algorithm and eventually to extract the key [BDL01].

Embedded devices have limited resources in terms of memory and space allocation, hence it is necessary to implement cryptographic algorithms with these constraints in mind. Elliptic curve cryptography (ECC) is suitable for lightweight protocols due to the small key length and memory requirements compared to equivalent RSA implementations. ECC is broadly used for digital signatures (ECSDA) and for establishment of a common secret key (ECDH). Sometimes ECC is also used for encryption (EC El-Gamal), but this is not a common practice, since symmetric cryptography offers faster algorithms for encrypting messages. By the time this thesis started, in 2013, the adoption of ECC in IT security systems was rising

and there was an emerging need for secure ECC real-world implementations. Although there were already many published papers on attacks, countermeasures and evaluations of ECC implementations, for instance

At the same time, the relevant research results for side-channel attacks on symmetric algorithms could not be compared to ECC evaluations. It is obvious that there was space for exploring new attack techniques and countermeasures in the asymmetric setting. And that was a big challenge, since asymmetric cryptography in general, and ECC in particular, provides slower operations than symmetric key algorithms. Therefore, the acquired traces that need to be collected and analyzed are very large. Moreover, some known attack techniques are not directly applicable to protocols such as ECDSA. For instance, DPA as proposed by Kocher et al. [KJJ99] does not apply to ECC protocols, where a scalar is used only once, like in ECDSA or in ephemeral Diffie-Hellman. The latter is incompatible with the requirement of DPA to collect many power traces of computations on the same secret data.

We overcame these challenges by using the appropriate equipment, by adapting the known attack techniques to our measurements and by exploring new ways of attacking/securing ECC implementations. The richness of the mathematical structures behind ECC, the various ways of representing a scalar and other algorithm-dependent features, created opportunities for many unique side-channel attacks exploiting those features. These techniques will be explained in the rest of this thesis.

1.2 Research Questions

There are various ways to implement ECC algorithms in embedded devices. In principle, the following techniques can lead to a secure implementation of an algorithm.

- The algorithm runs in constant time: Regarding cryptography, a constant time implementation means that the execution time does not depend on secret data. More precisely, there should be no branches on data that are supposed to be kept secret (exponent or scalar) and carries during modular multiplication should be handled in the same way as the results with no carries. This technique is important, as conditional branches and differences in the execution time according to the manipulated key bit are an easy target for attack and exploitation.

- The sensitive data are not used in clear: Masked (or blinded) implementations hide the sensitive data (usually the secret exponent or scalar). There are various ways to mask a value depending on the properties of the field that this value belongs to. For instance, there is Boolean, multiplicative, or additive masking. SPA cannot be applied to masked values, the adversary needs to be able to observe this value in two different time instances, and, therefore, he needs to apply DPA techniques.

- The secret keys are changed regularly: Secret keys should change in every execution of the algorithm, or every few executions. In cryptographic protocols that are used for key exchange, there should be two public keys, one that is long-term and verified by a certification authority, and one that is short-term (or ephemeral), it is signed by the long-term key for authentication, and it is used for one (or a few) session(s). In this way, an adversary who is able to capture one execution of the algorithm, is not able to perform SPA. If he is able to capture many executions of the algorithm, he could apply DPA, but if different keys are used, then he cannot recover them in a straight-forward way.

In most of real-world applications, at least one of the above mentioned techniques is not implemented correctly. This issue can be caused by careless design decisions or by speed reduction created by the adoption of many countermeasures, which makes them impractical to use.

In this context, the main research question that arises is *"Which practical and theoretically supported techniques can be developed to make ECC cryptographic implementations more side-channel resilient?"*. This question involves many parameters and requirements, in order to be answered. This fact motivates our research in the following, more concrete, subquestions:

1. What are the vulnerabilities of "secure" algorithms? Are there new attack techniques that would make it hard for a developer to implement secure ECC algorithms?

   There is a constant need for secure implementations. Algorithms that were considered secure 10 years ago might not be secure anymore, due to the discovery of new attack techniques. Thinking as an adversary and finding new vulnerabilities in a system is the first step towards a proposal for secure implementations.

2. Which countermeasures are efficient against newly proposed attacks? Can we propose new, efficient countermeasures to protect public-key algorithms?

   Every time a new attack technique shows up, there is a proposal of countermeasures that would thwart it. Countermeasures are expensive to implement in terms of time of execution and computational resources. It is, therefore, technically not possible to integrate many countermeasures in one implementation that has specific requirements for area, gates, power consumption and speed. ECC algorithms in particular started being broadly used in the last decade, after the relevant patents expired. They offered a compact and fast way to perform public-key protocols, and they found many applications to

embedded chips with power or size limitations, such as RFID tags. With the broad use of ECC and the challenges of software developers to implement big-number arithmetic in small-size circuits, a strong need emerged for secure ECC implementations. Countermeasures that were appropriate for RSA, for instance RSA-CRT, are not applicable to ECC. Others, such as the square-and-always-multiply, could be easily adapted for ECC, yielding the double-and-add-always algorithm. But still, the underlying mathematical structure and field operations in ECC offer the potential for applying new countermeasures, for instance working with unified formulas for addition and doubling can make some template and collision attacks hard to succeed.

3. Can we provide a systematic evaluation of countermeasures and choose a combination of them according to the security requirements?

   Practical evaluation of an ECC implementation using various countermeasures is obviously happening in side-channel labs. Evaluators have found various ways to attack ECC implementations, even in the presence of countermeasures. By experience acquired from many projects over the course of the years, software and hardware designers are capable to propose a combination of efficient countermeasures to their clients. However, the details of these designs and evaluations are not publicly available, and they would still be very useful for the research community, in order to validate the existing proposals, to establish the status of secure ECC algorithms and to promote new research directions in the field.

1.3 Organization of This Thesis

This thesis presents the results of the research performed, in order to answer the above mentioned research questions. More precisely, the chapters are organized as follows:

Chapter 2 gives an overview of Elliptic Curve Cryptography, covering aspects of the arithmetic of elliptic curves, various scalar multiplication algorithms and cryptographic protocols that use elliptic curves. The goal of this chapter is to provide background about different aspects of elliptic curves, which is necessary in order to comprehend the rest of the chapters.

Chapter 3 explores the side-channel attacks applicable to elliptic curve algorithms. After an introduction to the most known side-channel attack techniques, this chapter gives a broad literature overview of important attacks on elliptic curve implementations, and more specific on the scalar multiplication

operation, which is the most sensitive operation during the execution of elliptic curve protocols, since it manipulates the private key. A description of the Test Vector Leakage Assessment is also given with focus on adaptations for public-key algorithms.

Chapter 4 presents a new, elegant and efficient attack technique called *Online Template Attacks* (OTA). OTA is a novel attack technique that resides between horizontal and template attacks. This chapter introduces the theoretical aspects of OTA and shows how the attack applies to different scalar-multiplication algorithms by specific examples. A practical OTA is performed on double-and-add-always scalar multiplication on an ATMega card. The attack is generalized on Weierstrass curves of an implementation of mbedTLS on a Cortex-M4 micro-controller. Moreover, distinguishers from machine learning are applied during the template matching phase of OTA, in order to improve the success rates of the attack. A discussion on vulnerabilities of implementations and efficient countermeasures against OTA concludes this chapter.

Contribution of the author: The author established the theoretical foundations that would make the attack applicable to many scalar multiplication algorithms and developed the attack technique on which the practical experiments are based. This research resulted in two papers, namely "Online Template Attacks" in Indocrypt 2014 [BCP+14] and an extensive version of this work in the Journal of Cryptographic Engineering [BCP+17]. After the successful application of the power analysis OTA on the Edwards curve 25519, she investigated the wide applicability of the attack to other platforms and curves, resulting in the work of "Dismantling real-world ECC with Horizontal and Vertical Template Attacks" [DPN+16]. In the follow-up work from Özgen et al. [OPB16], the author proposed and co-supervised a Master thesis project on using distinguishers from machine learning (classification methods) together with OTA, which resulted in correct predictions on the scalar bit and a successful application of OTA. All these results are also included in the book chapter "Recent Developments in Side-Channel Analysis on Elliptic Curve Cryptography Implementations" in [PBM17].

Chapter 5 proposes algorithmic countermeasures to protect various scalar multiplication algorithms against known side-channel attacks. These countermeasures are based on a new technique called *Boolean XOR splitting*, that breaks the exponent or the scalar into Boolean shares and performs the corresponding algorithms using these shares. The security of the proposed algorithms

is verified theoretically using the Mutual Information (MI) framework and practically using the Test Vector Leakage Assessment (TVLA) methodology. The coauthors of the corresponding paper created the diagrams of MI and performed the practical TVLA experiments; both are included at the end of this chapter for the sake of completeness.

> **Contribution of the author:** The author contributed in the theoretical validation of the idea and more precisely in the application of Boolean XOR splitting in scalar multiplication algorithms for ECC. Applying the XOR splitting technique to elliptic curves needs special care, because of the point at infinity, as it is explained in Chapter 5. The author verified the correctness of the algorithms using Sage scripts. Moreover, she participated in the security evaluation of the algorithms against template attacks. The results of this research are included in the paper "Boolean XOR Splitting" [TPP18].

Chapter 6 explores the potential of the Residue Number System (RNS) as a numerical countermeasure against side-channel attacks on elliptic curves. Scalar multiplication, the key operation behind elliptic curve cryptography, is composed of arithmetic operations over finite fields. Thus, the introduction of RNS as a number system for arithmetic of the field elements can be a step towards increasing the side-channel resistance of cryptographic algorithms. However, this does not constitute enough protection against SCAs and fault attacks nor does it guarantee efficient implementations.

> **Contribution of the author:** This chapter is a collection of the results from the following published papers "Residue Number System as a Side-Channel and Fault Injection Attack Countermeasure in Elliptic Curve Cryptography" [FPBS16], "Secure and Efficient RNS Software Implementation for Elliptic Curve Cryptography" [FPS17] and "Practical Evaluation of Protected Residue Number System Scalar Multiplication" [PFPB19]. The author participated in the high-level design of secure variations of the RNS implementation, which can be found in the GitHub repository of Fournaris [Fou18b]. The main contribution of the author is the practical evaluation of the various RNS implementations using the Test Vector Leakage Assessment methodology and the Template Attacks adapted to the specific implementation and available platforms.

Chapter 7 concludes this thesis by summarizing the most important results and

proposing practical ways to achieve secure elliptic curve cryptographic implementations. Research directions and yet-undiscovered potentials for secure implementations that stem out of this thesis are also presented as future work.

Chapter 2

Elliptic Curve Cryptography

> Nothing can be truly replicated. Not a love, not a jewel, not a single line.
>
> *Patti Smith, M Train*

Public-key cryptography is the branch of cryptography that includes asymmetric algorithms used for key exchange, digital signatures, authentication and encryption. The principles of public-key agreement were established by Diffie and Hellman in the mid 70's [DH76] and provided new directions for the cryptographic community. The most important outcome of their research is the elimination of the need to establish a common encryption key prior to the communication phase, usually during an offline key agreement. Each user of a cryptographic protocol has in his possession a public and private key pair, which derive from the same mathematical object and which have a very interesting property. The public parameter can be used by everyone to perform encryption or verification of a signature, actions that are mathematically represented by an exponentiation or multiplication. However, inverting this operation, i.e. decrypting a message or creating a valid signature on a document, is a computationally hard problem and requires knowledge of the private key. Public-key algorithms have a variety of applications, such as network security protocols (SSL/TLS, SSH, IPsec), banking systems (PIN verification, authentication), communication applications (WhatsApp, Viber, Signal, Messenger), digital certificates and so on.

The most known and widely adopted public-key cryptosystem is RSA by Rivest, Shamir and Adleman [RSA78]. It is based on factorization and its security relies on the difficulty of extracting modular e-th roots when the factorization of the modulus is unknown. In this thesis we focus on Elliptic curve cryptosystems and we use the relevant mathematical notation, but the notation, and consequently the algorithms and countermeasures, can be easily transformed to multiplicative notation and applied to RSA.

Elliptic curves are mathematical objects with interesting algebraic and geometric properties that have been studied over a hundred years by mathematicians. Among the most fascinating applications of elliptic curves are algorithms for factoring integers, primality tests and the Taniyama–Shimura–Weil conjecture which is fundamental to proving Fermat's last theorem. This chapter provides preliminary knowledge about elliptic curve cryptography that is necessary to comprehend the results in the remainder of this thesis; it does not intend to be exhaustive. For a more comprehensive study of the topic, the interested reader is referred to the books [HMV03, CFA+05, Sil09, Sil94, Gal12].

Elliptic Curve Cryptography (ECC) was introduced in 1985 independently by Miller [Mil85] and Koblitz [Kob87]. It is broadly used for implementing asymmetric cryptographic protocols (public-key encryption and digital signatures); it is preferred for embedded devices due to the small key length and memory requirements compared to equivalent RSA implementations. For instance, a 256-bit field curve provides a security level of 128 bits, which is roughly equivalent to a 3248-bit RSA key (see [BCC+12] for more details).

The security of elliptic curve cryptosystems relies on the difficulty of solving the Discrete Logarithm problem on elliptic curves defined over finite fields of large characteristic or binary fields. In this chapter, the three basic aspects of ECC are presented, namely the arithmetic of elliptic curves in Section 2.1, scalar multiplication algorithms on elliptic curves in Section 2.2 and cryptographic protocols in Section 2.3.

2.1 Arithmetic of Elliptic Curves

Elliptic curve arithmetic is performed in terms of the underlying field operations. For ECC, the curves are defined over prime or binary fields.

Definition 2.1.1. A *prime field* $\mathbb{F}_p$, where $p > 3$ is a prime, is a finite field with p elements, which determines it uniquely up to isomorphism. It is represented by the set of integers $\{1, ..., p\}$ with the addition and multiplication operations performed modulo p.

Definition 2.1.2. A *binary field* of order 2^m consists of all the binary polynomials of degree at most $m-1$, $\mathbb{F}_{2^m} = \{a_{m-1}z^{m-1} + ... + a_1 z + a_0 : a_i \in \mathbb{F}_2\}$. Addition, multiplication and inversion of field elements are performed modulo an irreducible binary polynomial $f(z)$ of degree m.

Definition 2.1.3. The *characteristic* of a field $\mathbb{F}$ is defined as the smallest number $m > 0$ such that $m \cdot e = e + e + \cdots + e = 0$, where e is the unit element of $\mathbb{F}$.

Given these introductory definitions, we can proceed with defining the arithmetic properties of elliptic curves used in cryptography.

2.1.1 Coordinate Systems

An elliptic curve $\mathcal{E}$ over the finite field *K*, denoted as $\mathcal{E}_K$, can be defined in terms of solutions (x, y) to one of the equations defined in Section 2.1.2. The pairs that verify these equations represent the *affine coordinates* of a point over the curve $\mathcal{E}$. From the addition rules on an elliptic curve, the necessary operations are addition, multiplication and inversion over *K*. Inversion is the most expensive operation and can be avoided by using other types of coordinate systems for the points $\boldsymbol{P} = (x, y)$. We hereby present the most commonly used coordinates systems that can be found in cryptographic implementations of elliptic curve protocols.

Projective coordinates
In the projective coordinate system, each point $\boldsymbol{P} = (x, y)$ is represented by three coordinates (X, Y, Z), where $x = \frac{X}{Z}, y = \frac{Y}{Z}$, with $Z \neq 0$.

Jacobian coordinates
In the Jacobian coordinates system, each point $\boldsymbol{P} = (x, y)$ is represented also by three coordinates (X, Y, Z), with $x = \frac{X}{Z^2}, y = \frac{Y}{Z^3}, Z \neq 0$.

López-Dahab coordinates
In the López-Dahab system, the relation for the point (X, Y, Z) is $x = \frac{X}{Z}, y = \frac{Y}{Z^2}$ with $Z \neq 0$.

2.1.2 Forms of Elliptic Curves

There are several types of elliptic curves defined by their curve equation. Below we will treat some commonly used forms.

Weierstrass curves

Definition 2.1.4. An elliptic curve defined over a field K is defined by the Weierstrass equation

$$\mathcal{E}_K : y^2 + \alpha_1 xy + \alpha_3 y = x^3 + \alpha_2 x^2 + \alpha_4 x + \alpha_6 \,. \tag{2.1}$$

Together with the point at infinity $\mathcal{O}$, the set $(\mathcal{E}_K \cup \mathcal{O}, +)$ forms an Abelian group with neutral element $\mathcal{O}$.

Detailed proofs that the three axioms for abelian groups hold for Equation 2.1 can be found in [Sil09, Chapter 3] and of [Ste09, Chapter 6].

When the characteristic of the field $char(K)$ is not 2 or 3, then the general Weierstrass form can be simplified to

$$\mathcal{E}_K : y^2 = x^3 + \alpha x + \beta. \tag{2.2}$$

In the following, it is assumed that $char(K) \neq 2, 3$. Adding the points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ gives a third point on the curve, namely $P + Q = (x_3, y_3)$ according to the formulæ

$$\begin{cases} x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= \lambda(x_1 - x_3) - y_1, \end{cases}$$

with $\lambda = \dfrac{y_1 - y_2}{x_1 - x_2}$ if $P \neq \pm Q$ and $\lambda = \dfrac{3x_1^2 + \alpha}{2y_1}$ if $P = Q$.

For points represented in Jacobian coordinates $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$, the addition of P and Q with $P \neq \pm Q$ is $P + Q = (X_3, Y_3, Z_3)$ with

$$X_3 = F^2 - E^3 - 2BE^2, Y_3 = F(BE^2 - X_3) - DE^3, Z_3 = Z_1 Z_2 E, \quad (2.3)$$

where $A = X_1 Z_2^2$, $B = X_2 Z_1^2$, $C = Y_1 Z_2^3$, $D = Y_2 Z_1^3$, $E = A - B$, $F = C - D$ [BL]. Jacobian addition needs $12\mathbf{M} + 4\mathbf{S}$, with $\mathbf{M}$ and $\mathbf{S}$ the number of multiplications and squarings over K, respectively. Point doubling can be performed very efficiently with only $3\mathbf{M} + 6\mathbf{S}$ using the formulæ

$$X_3 = B^2 - 2A, Y_3 = B(A - X_3) - Y_1^4, Z_3 = Y_1 Z_1, \quad (2.4)$$

where $A = X_1 Y_1^2, B = \dfrac{1}{2}(3X_1^2 + \alpha Z_1^4)$.

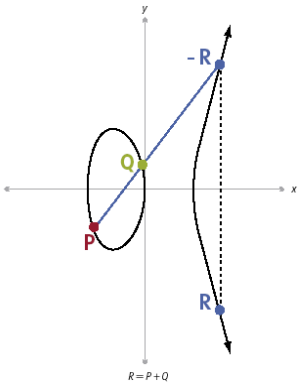

Figure (2.1) Point addition on an elliptic curve

Weierstrass curves are standardized and widely used in cryptography [AX98, Res00, NIS13, BSI10]. However, they had a main drawback regarding their side-channel resistance; namely their addition formulæ were incomplete and not unified. An elliptic curve addition law is *complete* if it correctly computes the sum of any two points in the group without exceptions. The addition law is *unified* if addition and doubling of points can be computed by the same formulæ. As it is obvious from the previous formulæ, addition and doubling are handled differently and the point at infinity gives an exceptional case. In [BL95], Bosma and Lenstra presented complete formulæ for Weierstrass curves, which had an exceptional case for the pair of points (P, Q) if and only if $P - Q$ is a point of order two. In [RCB16], Renes, Costello and Batina presented complete addition formulæ for prime order

elliptic curves $E/\mathbb{F}_{\text{ıı}} : y^2 = x^3 + \alpha x + b$ with $q \geq 5$, which require only 12 field multiplications $12\mathbf{M}$. [1] The complete addition formulæ using homogeneous representation of a point are

$$\begin{cases} X_3 &= (X_1Y_2 + X_2Y_1)(Y_1Y_2 - \alpha(X_1Z_2 + X_2Z_1) - 3bZ_1Z_2) \\ & \quad -(Y_1Z_2 + Y_2Z_1)(\alpha X_1X_2 + 3b(X_1Z_2 + X_2Z_1) - \alpha^2 Z_1Z_2), \\ Y_3 &= (Y_1Y_2 + \alpha(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2)(Y_1Y_2 - \alpha(X_1Z_2 + X_2Z_1) \\ & \quad -3bZ_1Z_2) + (3X_1X_2 + \alpha Z_1Z_2)(\alpha X_1X_2 + 3b(X_1Z_2 + X_2Z_1) \\ & \quad -\alpha^2 Z_1Z_2), \\ Z_3 &= (Y_1Z_2 + Y_2Z_1)(Y_1Y_2 + \alpha(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2) \\ & \quad +(X_1Y_2 + X_2Y_1)(3X_1X_2 + \alpha Z_1Z_2) \end{cases}$$

The completeness property of the addition law offers the possibility to have secure ECC implementations. By removing the "branching" to handle exceptional cases, some side-channel attack vulnerabilities relying on handling exceptional cases can be prevented. It is trickier, but still possible, to achieve secure ECC implementations also without complete addition formulæ, but the difference between addition and doubling operations should be handled carefully. The way the multiplication is handled in the underlying field operations, and more specifically the handling of carries, is very crucial for side-channel leakages as we will see in the next chapters.

Edwards curves

Edwards curves, introduced by Edwards in [Edw07], were the first curves shown to have a complete addition law. Applications of Edwards and twisted Edwards curves in cryptography are extensively studied by Bernstein and Lange [BL07, BL09] and Hisil et al. [HWCD08]. An Edwards curve is defined over a field K with $char(K) \neq 2$ by the equation:

$$\mathcal{E}_d : y^2 + x^2 = 1 + dx^2y^2, \tag{2.5}$$

where $d \in K\backslash\{0, 1\}$. The Edwards addition law for two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, in affine coordinates, is given by the following formulæ:

$$P + Q = (x_3, y_3) = (\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2}). \tag{2.6}$$

This addition law is unified, i.e. the same formula can be used for both addition and doubling without exceptional cases. The neutral element is the point $(0, 1)$. If d is not a square, then the addition law is complete and there are no exceptional cases for the neutral element.

[1] For the overview on elliptic curves in this section, we omit counting the multiplications by a constant (M_α) and the additions $\mathbf{A}$, which are used for extensive comparison results in some publications.

Twisted Edwards curves, introduced in [BBJ+08], are a generalization of Edwards curves with the form $E_{\alpha,d} : y^2 + \alpha x^2 = 1 + dx^2y^2$. The addition law for twisted Edwards curves is a generalization of Equation (2.6):

$$P + Q = (x_3, y_3) = (\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - \alpha x_1x_2}{1 - dx_1x_2y_1y_2}). \tag{2.7}$$

The cost of addition and doubling on Edwards curves depends on the form of the curve and the coordinates chosen by the developer. An overview of all types of curves and coordinates is given in the Explicit Formulas Database [BL]. The implementation of Bernstein et al. in [BBJ+08] uses inverted twisted Edwards coordinates and needs $9\mathbf{M} + 1\mathbf{S}$ for addition and $3\mathbf{M} + 4\mathbf{S}$ for doubling. The most efficient implementation of twisted Edwards curves is given by Hisil et al. [HWCD08] using $8\mathbf{M}$ for addition with suitably selected curve constants.[2]

Montgomery curves
In [Mon87], P. L. Montgomery defined the following form of elliptic curves over finite fields of odd characteristic:

$$\mathcal{E}_M : By^2 = x^3 + Ax^2 + x, \quad B(A^2 - 4) \neq 0. \tag{2.8}$$

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on $\mathcal{E}_M$. Then, the point $P_3 = (x_3, y_3) = P_1 + P_2$ can be calculated using the following formulæ:
Addition formulæ $(P_1 \neq \pm P_2)$

$$\begin{cases} \lambda &= (y_2 - y_1)/(x_2 - x_1) \\ x_3 &= B\lambda^2 - A - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{cases}$$

Doubling formulæ $(P_1 = P_2, y_1 \neq 0)$

$$\begin{cases} \lambda &= (3x_1^2 + 2Ax_1 + 1)/(2By_1) \\ x_3 &= B\lambda^2 - A - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{cases}$$

Montgomery arithmetic is very efficient with additional speed-up by computing only (X, Z) coordinates of intermediate points [Sta03]. We set $(x, y) = (X/Z, Y/Z)$ and present the operations in projective coordinates, as described in [Mon87]. We note here that the point $nP = (X_n, Y_n, Z_n)$ is the n-times multiple of the point $P = (X, Y, Z)$. The addition and doubling formulæ for $(m+n)P = mP + nP$ are as follows:

[2] The developer can choose to use different formulæ for addition and doubling in twisted Edwards curves for extra efficiency in the implementation or unified formulæ for resistance against side-channel attacks.

Addition formulæ $(m \neq n)$

$$\begin{cases} X_{m+n} &= Z_{m-n}[(X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n)]^2 \\ Z_{m+n} &= X_{m-n}[(X_m - Z_m)(X_n + Z_n) - (X_m + Z_m)(X_n - Z_n)]^2 \end{cases}$$

Doubling formulæ $(m = n)$

$$\begin{cases} 4X_nZ_n &= (X_n + Z_n)^2 - (X_n - Z_n)^2 \\ X_{2n} &= (X_n + Z_n)^2(X_n - Z_n)^2 \\ Y_{2n} &= (4X_nZ_n)((X_n - Z_n)^2 + ((A+2)/4)(4X_nZ_n)) \end{cases}$$

In [IMT], it is shown that the loop iteration in the Montgomery power ladder 2.2.4 using (X, Z) coordinates is performed in only $11\mathbf{M} + 4\mathbf{S}$. Moreover, as presented in [OKS00], the number of additions and doublings in scalar multiplications on Montgomery form elliptic curves only depends on the bit length of the key and not on the bit patterns or the bit itself at a certain position of the scalar.

Hessian curves

A Hessian curve over a field *K* is defined by the cubic equation

$$\mathcal{E}_K : x^3 + y^3 + z^3 = dxyz, \tag{2.9}$$

where $d \in K$ and $d^3 \neq 27$. Hessian and twisted Hessian curves are interesting for cryptography due to their small cofactor 3 and their side-channel resistance [HWCD09, JQ01, BCKL15]. Moreover, the Hessian addition formulæ (also called Sylvester formulas) can be used for doubling, a fact that provides a form of unification.

In [FJ10], Farashahi and Joye presented efficient unified formulæ for *generalized* Hessian curves:

$$\mathcal{E}_K : x^3 + y^3 + cz^3 = dxyz, \tag{2.10}$$

where $c, d \in K$, $c \neq 0$ and $d^3 \neq 27c$. The unified formulæ are complete for certain parameter choices. More precisely, the group of *K*-rational points on a generalized Hessian curve has complete addition formulæ, if and only if c is not a cube in *K*. The fastest known addition formulæ on binary elliptic curves with $9\mathbf{M} + 3\mathbf{S}$ for extended projective coordinates and $8\mathbf{M} + 3\mathbf{S}$ for mixed affine-projective addition are presented in [FJ10]. The sum of two points P, Q represented in extended projective coordinates by $(X_i : Y_i : Z_i : A_i : B_i : C_i : D_i : E_i : F_i)$, where $A_i = X_i^2, B_i = Y_i^2, C_i = Z_i^2, D_i = X_iY_i, E_i = X_iZ_i, F_i = Y_iZ_i$ and for $i = 1, 2$, is the point $P + Q = (X_3 : Y_3 : Z_3 : A_3 : B_3 : C_3 : D_3 : E_3 : F_3)$ with

$$\begin{cases} X_3 = cC_1F_2 + D_1A_2 \\ Y_3 = B_1D_2 + cE_1C_2 \\ Z_3 = A_1E_2 + F_1B_2, \end{cases}$$

with $A_3 = X_3^2$, $B_3 = Y_3^2$, $C_3 = Z_3^2$, $D_3 = X_3Y_3$, $E_3 = X_3Z_3$, $F_3 = Y_3Z_3$.

The complete addition formulæ for twisted Hessian curves of cofactor 3 in [BCKL15] give the fastest results for prime-field curves with $8.77\mathbf{M}$ for certain curve parameters. Bernstein et al. [BCKL15] ran the scalar multiplication algorithm for 10000 random 256-bit scalars, i.e., integers between 2^{255} and $2^{256} - 1$, using as input the costs of twisted Hessian operations and got $8.77\mathbf{M}$ per bit as the average cost of the resulting addition chain.

2.2 Scalar Multiplication Algorithms

Elliptic curve operations are used for cryptographic protocols such as the Elliptic Curve Digital Signature Algorithm (ECDSA) for digital signatures, Elliptic Curve ElGamal as an encryption/decryption scheme, and Elliptic Curve Diffie-Hellman (ECDH) as a key exchange scheme. The main operation in all those protocols using ECC is *scalar multiplication* of a point $\boldsymbol{P}$ on a curve with an integer k.

Computing the result of a scalar multiplication on an elliptic curve can be done in a similar way as exponentiation in RSA. A simple and efficient algorithm is binary scalar multiplication, where an n-bit scalar k is written in its binary form $(k_0, k_1, ..., k_{n-1})_2$ with $k = \sum_i k_i 2^i, k_i \in \{0, 1\}$. A binary algorithm processes a loop, scanning the bits of the scalar (from the most significant bit to the least significant one, or the other way around) and performing a point doubling only if the current bit is 0 or a doubling and an addition if the bit is 1.

Scalar multiplication algorithms take as input a point $\boldsymbol{P}$ in affine or projective coordinates and the scalar k. The result is the point $[k]\boldsymbol{P}$ on the curve. During the execution of the algorithm, mixed coordinates can be used for an additional speed-up [CMO98].

Scalar multiplication is a sensitive operation from a security point of view, since it manipulates the secret key k and returns the result according to the bits of the key. Naïve implementations of scalar multiplication with `if`−statements are subject to side-channel attacks, and more precisely timing attacks. Coron's randomization countermeasures of point or scalar, as presented in [Cor99], can thwart timing or DPA attacks, but not SPA attacks. SPA leakage is present when there is a difference in operation flow between only doubling or doubling-and-addition. Point additions and point doublings may result in different leakage patterns and since these operations are key dependent, the key could be retrieved quite easily. SPA resistant algorithms based on unified formulæ are *regular* algorithms, which perform a constant operation flow regardless of the scalar value. A nice overview of fast and regular scalar multiplication algorithms is given in [Riv11]. In this section, we present the

most broadly used regular scalar multiplication algorithms. It is important to note here, that this is not an exhaustive presentation of scalar multiplication algorithms; we focus only on the algorithms used by the implementations that we attacked and protected in terms of this research.

2.2.1 Left-to-right double-and-add Algorithm

The straightforward implementation of scalar multiplication on an elliptic curve involves repeated doubling and addition operations. When the bit of the scalar is 0, a doubling is performed, and when the bit is 1 a doubling is followed by addition. In the following algorithm, we show how such an implementation could be written.

Algorithm 1: The left-to-right double-and-add algorithm

Input: $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: $\boldsymbol{Q} = [k]\boldsymbol{P}$

1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$;
2 **for** $i \leftarrow n-2$ **down to** *0* **do**
3 $\quad \boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$;
4 $\quad$ **if** $k_i = 1$ **then**
5 $\quad\quad \boldsymbol{R}_1 \leftarrow \boldsymbol{R}_0 + \boldsymbol{P}$;
6 $\quad$ **end**
7 **end**
8 **return** $\boldsymbol{R}_0$

In the next chapter, we show the vulnerabilities caused by the conditional statement `if` $k_i = 1$.

2.2.2 Left-to-right double-and-add-always Algorithm

The double-and-add-always algorithm was initially proposed by Coron in [Cor99] as a first attempt to avoid `if`−statements and therefore prevent the identification of different operations. The algorithm performs a point doubling followed by a point addition in a `for` loop, scanning the scalar bits from the most significant to the least significant one. Both operations are performed in every loop and according to the key bit, the final assignment to $\boldsymbol{R}_0$ will be either $\boldsymbol{R}_0$ or $\boldsymbol{R}_1$. There are no conditional statements in the algorithm, but there is one key-dependent assignment, which can leak secret information. Another important remark is that $\boldsymbol{R}_0$ is initialized by $\boldsymbol{P}$ instead of $\mathcal{O}$, in order to avoid exceptional cases given by the point at infinity.

Algorithm 2: The left-to-right double-and-add-always algorithm [Cor99]

Input: $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: $\boldsymbol{Q} = [k]\boldsymbol{P}$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$;
for $i \leftarrow n-2$ **down to** *0* **do**
  $\boldsymbol{R}_0 \leftarrow 2\boldsymbol{R}_0$;
  $\boldsymbol{R}_1 \leftarrow \boldsymbol{R}_0 + \boldsymbol{P}$;
  $\boldsymbol{R}_0 \leftarrow \boldsymbol{R}_{k_i}$;
end
return $\boldsymbol{R}_0$

2.2.3 Right-to-left double-and-add-always Algorithm

The binary right-to-left double-and-add-always algorithm proposed by Joye [Joy07] is shown below as Algorithm 3. The steps of the algorithm are similar to Algorithm 2 with the following differences:

- The bits of the scalar are scanned from the least significant to the most significant one.
- Two temporary registers are used instead of three and they are both effectively used, without any dummy operations.

Algorithm 3: Binary right-to-left double-and-add-always algorithm [Joy07]

Input: $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: $\boldsymbol{Q} = [k]\boldsymbol{P}$

$\boldsymbol{R}_0 \leftarrow \mathcal{O}$;
$\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
for $i \leftarrow 0$ **up to** *n-1* **do**
  $b \leftarrow 1 - k_i$;
  $\boldsymbol{R}_b \leftarrow 2\boldsymbol{R}_b$;
  $\boldsymbol{R}_b \leftarrow \boldsymbol{R}_b + \boldsymbol{R}_{k_i}$;
end
return $\boldsymbol{R}_0$

Similar to Algorithm 2, there are no conditional statements in this algorithm, but there is a key-dependent assignment, which can be vulnerable to attacks. However, there are several attacks that can be mounted on the left-to-right, but not on the right-to-left algorithm (for instance the Doubling attack, described in Chapter 3, is only applicable on the left-to-right algorithm).

2.2.4 Montgomery Power Ladder

The Montgomery Power Ladder (MPL), initially presented by Montgomery [Mon87] as a way to speed up scalar multiplication on elliptic curves, is one of the most challenging algorithms for simple side-channel analysis due to its natural regularity of operations. This algorithm is used as the primary secure and efficient choice for resource-constrained devices. A comprehensive security analysis of the MPL, given by Joye and Yen in [JY02], showed that the regularity of the algorithm makes it intrinsically secure against a large variety of implementation attacks (SPA, some fault attacks, etc.). The MPL is described in Algorithm 4. For a specific choice of projective coordinates, as described in Section 2.1.2 and in [Sta03], one can do computations with only X and Z coordinates, which makes this option more memory efficient than other algorithms.

Algorithm 4: The Montgomery Ladder

Input: $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: $\boldsymbol{Q} = [k]\boldsymbol{P}$

$\boldsymbol{R}_0 \leftarrow \mathcal{O}$;
$\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
for $i \leftarrow n-1$ **down to** *0* **do**
    $b \leftarrow 1 - k_i$;
    $\boldsymbol{R}_b \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_1$;
    $\boldsymbol{R}_{k_i} \leftarrow 2 \cdot \boldsymbol{R}_{k_i}$;
end
return $\boldsymbol{R}_0$

2.2.5 Side-Channel Atomicity

Side-channel atomicity is an SPA countermeasure proposed by Chevallier-Mames et al. [CMCJ04], in which individual operations are implemented in such a way that they have an identical side-channel profile (e.g. for any branch and any key-bit related subroutine). In short, it is suggested in [CMCJ04] that the point doubling and addition operations are implemented such that the same code is executed for both operations. This renders the operations indistinguishable by simply inspecting a suitable side-channel. One could, therefore, implement a point multiplication as described in Algorithm 5.

We note three side-channel equivalent instructions in the `while`-loop, meaning that they have the same order of operations and therefore produce indistinguishable type of leakage. For instance, we can assume that the dummy instruction $i \leftarrow i - 0$ is side-channel equivalent to the instruction $i \leftarrow i - 1$. Point doubling will be

Algorithm 5: Side-Channel Atomic double-and-add algorithm [CMCJ04]

Input: $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: $\boldsymbol{Q} = [k]\boldsymbol{P}$

$R_0 \leftarrow \mathcal{O}$; $R_1 \leftarrow \boldsymbol{P}$; $i \leftarrow n-1$;
$m \leftarrow 0$;
while $i \geq 0$ **do**
  $\boldsymbol{R}_0 \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_m$;
  $m \leftarrow m \oplus k_i$;
  $i \leftarrow i - \neg m$;
end
return $\boldsymbol{R}_0$

performed if $m = 0$ and depending on the current scalar bit k_i, the variable i will decrement or not. It is worth mentioning that this protected algorithm requires on average $1.5 \cdot n$ multiplications, which is the same complexity as the unprotected double-and-add algorithm.

There are certain choices of coordinates and curves for which this approach can be deployed by using unified or complete addition formulæ for the group operations. The unified and complete formulæ of Weierstrass, Edwards and Hessian curves are described in Section 2.1.2.

2.2.6 Nonadjacent Form

The *nonadjacent form (NAF)* of a scalar $k = \sum_{i=0}^{n} k_i 2^i$ is a representation with $k_i \in \{-1, 0, 1\}$ and $k_i k_{i+1} = 0$ for all $i \geq 0$. Since 1960, Reitwiesner has shown that every integer $m \in \mathbb{Z}$ has exactly one NAF [Rei60] and Gordon proved that the NAF(m) has the fewest nonzero coefficients of any signed binary expansion of m, since the expected number of nonzeros in a length n NAF is $n/3$ [Gor98] on average. This fact allows further optimizations in the efficiency of scalar multiplication using NAF [JT01a]. From an efficiency point of view, using NAF for the representation of the scalar in the double-and-add algorithm is good, because it would require m doublings and $m/3$ additions or subtractions on the curve on average.

The use of NAF representation for speeding up the computations on elliptic curves was initially proposed by Morain and Olivos [MO90], mainly for improving factorization and primality tests. Inverses on elliptic curves can be computed "for free", since the inverse $-\boldsymbol{P}$ of a point $\boldsymbol{P} = (x, y)$ geometrically represents the symmetrical point with respect to the x-axis with coordinates $(x, -y)$. Therefore, using addition-subtraction chains for scalar multiplication and allowing negative digits in the binary representation of the scalar is an efficient choice for additive groups.

Following the description of Ha and Moon [CHJM03], a scalar k with an n-bit binary representation can be NAF decoded into an $(n+1)$-bit integer d; the number of subsequent additions-subtractions in the scalar multiplication is equal to the number of non-zero bits of d. The scalar multiplication with NAF decoded scalar can be computed as follows:

Algorithm 6: Scalar multiplication with NAF algorithm [CHJM03]

Input: $\boldsymbol{P}$, $n+1$-bit integer $d = (d_n, d_{n-1}, \ldots, d_0)$,s.t.$d_i \in \{-1, 0, 1\}$
Output: $\boldsymbol{Q} = d \cdot \boldsymbol{P}$

```
1  R ← 0;
2  for i ← x down to 0 do
3  |   R ← 2R ;
4  |   if d_i = 1 then
5  |   |   R ← R + P ;
6  |   end
7  |   if d_i = −1 then
8  |   |   R ← R − P ;
9  |   end
10 end
11 return R
```

The complexity of Algorithm 6 is $n/3$ additions and $n+1$ doublings on average. Because of the obvious efficiency on computing a scalar multiplication over elliptic curves, the NAF representation and different variations of it with window methods has been extensively examined in the cryptographic community. Ha and Moon [CHJM03] proposed a randomized version of the signed-scalar representation, in order to protect the implementation against certain side-channel attacks. They insert a n-bit long random number r, whose bits have the value 0 or 1 with equal probability $1/2$. As noted in their paper, the number of non-zero digits in the randomized version of the algorithm is on average $n/2$, which is the same as in the binary case, but a bit slower compared to the unprotected version of the NAF algorithm that needs on average $n/3$ additions. Of course the random number generator cost (either in hardware or in software) should also be taken into consideration when choosing which form of NAF algorithm is suitable for an application.

2.2.7 Fixed-base Comp Methods

In the case that the point $\boldsymbol{P}$ is fixed, then the scalar multiplication $[k]\boldsymbol{P}$ can be accelerated by precomputing multiples of $\boldsymbol{P}$, storing them in memory and retrieving them when necessary. For instance, precomputing the multiples of $\boldsymbol{P}$, where the scalar is $k = \{2, 2^2, 2^3, ...\}$ could eliminate the costs of doubling in the scalar multiplication

routine. Section 3.3.2 of [HMV03] gives a nice analysis of this technique, which was presented by López and Dahab [LD99] and it is based on the adaptation of the exponentiation method of Lim and Lee [LL94] for binary fields.

We present here a simplified version of the fixed-base comb multiplier by Brickell et.al [BGMW92], which is one of the fastest scalar multiplication methods available. As it requires intense precomputations, it is only used for multiplications with a fixed point, which is in most cases a generator of an elliptic curve (sub-)group.

The following algorithm shows the fixed-base comb method as presented in [HMV03].

Algorithm 7: Fixed-base comb method [HMV03, Chapter 3]

Input: $\boldsymbol{P}$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$, window width w, $d = \lceil n/w \rceil$
Output: $\boldsymbol{Q} = [k]\boldsymbol{P}$

1 Precompute $[a_{w-1}, ..., a_1, a_0]\boldsymbol{P}$ for all $(a_{w-1}, ..., a_1, a_0)$;
2 $k = K^{w-1}||...||K^1||K^0$;
3 $\boldsymbol{Q} \leftarrow \mathcal{O}$;
4 **for** $i \leftarrow d-1$ **down to** *0* **do**
5 $\quad \boldsymbol{Q} \leftarrow 2\boldsymbol{Q}$;
6 $\quad \boldsymbol{Q} \leftarrow \boldsymbol{Q} + [K_i^{w-1}, ..., K_i^1, K_i^0]\boldsymbol{Q}$;
7 **end**
8 **return** $\boldsymbol{Q}$

For this adapted method of Lim and Lee [LL94], the binary representation of k is first padded on the left with $dw - n$ zeros, where w is the window width. Let $(K_{d-1}, ...K_1, K_0)$ be the base 2^w representation of k. Then, k is divided into w bit strings, each of length d, so that $k = K^{w-1}||...||K^1||K^0$. The K^j bit strings of length d are written as rows of an array, whose columns are processed one by one. Brickell et.al [BGMW92] proposed to use partial sums of elements of this array, for instance $Q_j = \sum_{i:K_i=j} 2^{wi}\boldsymbol{P}$ for each j. In this way, the scalar multiplication becomes:

$$\begin{aligned}[k]\boldsymbol{P} &= \sum_{i=0}^{d-1} K_i(2^{wi}\boldsymbol{P}) = \sum_{j=1}^{2^w-1}(j\sum_{i:K_i=j}[2^{wi}]\boldsymbol{P}) = \sum_{j=1}^{2^w-1} jQ_j \\ &= Q_{2^w-1} + (Q_{2^w-1} + Q_{2^w-2}) + ... + (Q_{2^w-1} + Q_{2^w-2} + ... + Q_1)\end{aligned} \tag{2.11}$$

The values a_i represent all the possible bit strings of length w. If we set $[a_{w-1}, ..., a_0]$ $= a_{w-1}2^{(w-1)d} + ... + a_1 2^d + a_0$ and we precompute $2^{d/2}[a_{w-1}, ..., a_0]\boldsymbol{P}$, as in step 1 of Algorithm 7, we can accelerate the computation. In this way, the partial sums in Equation 2.11 can be computed faster and the scalar multiplication is then calculated as written in the `for`-loop of the algorithm.

This algorithm is constant-time and it is used by mbedTLS [mbe], in order to provide security for embedded systems. However, as we show in Chapter 4, it is

vulnerable to certain types of template attacks, even if some countermeasures are applied.

2.3 Elliptic Curve Protocols

As mentioned in Section 2.2, ECC primitives are used for cryptographic protocols such as the Elliptic Curve Digital Signature Algorithm (ECDSA) for digital signatures, the equivalent signature scheme for Edwards curves EdDSA, Elliptic Curve ElGamal as an encryption scheme, and Elliptic Curve Diffie-Hellman (ECDH) as a key agreement scheme. In this section, the details of those protocols are described.

In general, the use of elliptic curves in cryptography relies on the difficulty of solving the *Discrete Logarithm Problem*, so-called ECDLP, in the finite field where the elliptic curve is defined. The basic Discrete Logarithm Problem requires to find k where $x^k = y$ and x, y belong to the same cyclic group G with x being a generator of the group. The elliptic curve version requires to find a scalar k that satisfies the equation $\boldsymbol{Q} = [k]\boldsymbol{P}$, where points $\boldsymbol{P}, \boldsymbol{Q}$ belong to a set of points G on an elliptic curve. Actually, one works in the cyclic subgroup generated by G, where G is a point of large prime order $\mathrm{ord}(n)$ to avoid trivial attacks. This problem is known to be computationally difficult and all currently known algorithms to solve the problem are exponential. As mentioned earlier, the security level achieved using elliptic curves is higher for the same key size, for instance 256-bit ECC is considered to be equivalent to 3072-bit RSA achieving 128-bit security.

2.3.1 Elliptic Curve Diffie-Hellman

Elliptic Curve Diffie-Hellman (ECDH) is a variation of the original Diffie-Hellman key agreement protocol [DH76], where the multiplicative group of integers modulo a prime p is replaced by the additive group of points on an elliptic curve. As with the original protocol, ECDH is used when two parties A and B want to agree on a shared secret key over a public channel. It is common to use an asymmetric algorithm to establish the shared secret and continue the encrypted communication using a symmetric algorithm.

For the establishment of a shared secret with ECDH, the following steps are necessary:

1. Party A will generate a private key d_A and a public key $\boldsymbol{Q}_A = [d_A]G$ (where G is the generator for the curve $\mathcal{E}$).

2. Similarly, party B has his private key d_B and a public key $\boldsymbol{Q}_B = [d_B]G$.

3. Party B has to send his public key to A, in that way A can calculate $[d_A]\boldsymbol{Q}_B = [d_A \cdot d_B]G$, and the same holds for party B when he gets the public key of A.

Before the calculation of the shared key, both parties have to verify that the points they received belong indeed on the curve and they are of large order.

4. The shared secret is the x co-ordinate of the calculated point $[d_A \cdot d_B]G$.

This protocol is simple and it can provide security in a straightforward way. An eavesdropper over the public channel can know only $\boldsymbol{Q}_A$ and $\boldsymbol{Q}_B$, from which he can not derive the shared secret.

2.3.2 Elliptic Curve ElGamal

Elliptic Curve ElGamal is the equivalent of ElGamal encryption-decryption protocol using elliptic curves over a finite field. Following the definition of [Kob87], let $q = p^{n'}$ be a large integer with $n' \in \mathbb{Z}$, $\mathcal{E}$ defined over $\mathbb{F}_q$ and $\mathrm{ord}(\mathcal{E}) = n$. The public parameters of the cryptosystem are $\mathcal{E}, q, n$ and a function $f : m \mapsto \boldsymbol{P}_m$, which maps messages to points on the curve.

Suppose that party A wants to send an message m encrypted to party B. The following steps should be followed for an ElGamal encryption protocol:

1. A chooses a secret value $x \in_R [1, n-1]$ such that $0 < x < n-1$ and $\gcd(x, n) = 1$, and transmits $Y = [x]G$ to B.

2. Party B chooses a random k with $k \in_R [1, n-1]$ and creates the ciphertext $(C_1, C_2) = ([k]G, kY + P_m)$.

3. A can decrypt the ciphertext by calculating $C' = xC_1$ and $P_m = C_2 - C' = k([x]G) + P_m - x([k]G)$.

4. The original message m is the result of $f^{-1}(P_m)$.

2.3.3 Elliptic Curve Digital Signature Algorithms

Generating and verifying a signature using elliptic curves is equivalent to the original Digital Signature Algorithm (DSA). However, the high levels of security that can be achieved by using smaller key size compared to DSA, led ECDSA and similar signature systems like EdDSA and Ed25519 to be widely adopted. ECDSA, as presented in [JMV01] is standardized by several organizations (it appears in NIST FIPS 140-2 [FIP01], ANSI X9.62, IEEE 1363-2000 and ISO/IEC 15946-2 standards) and used by OpenSSL [Fou18a], Bitcoin [Nak09] and other cryptocurrencies.

Each party participating in the signature protocol possesses a key-pair, the private key d_i and the public key $Q_i = [d_i]G, i \in \{A, B\}$, for parties A and B, where G is the elliptic curve base point, a generator of the elliptic curve $\mathcal{E}$ with large prime order $\mathrm{ord}(n)$. The parameters $(\mathcal{E}, G, n)$ are public parameters of the system, on which both parties have agreed.

2.3.3.1 ECDSA Signature Generation

Let us assume that party A wants to send m signed to party B.

1. First, A calculates $h = hash(m)$, where $hash()$ is a secure cryptographic hash function.
2. A selects randomly $k \in_R [1, n-1]$ and calculates $(x_1, y_1) = [k]G$.
3. The signature is the tuple $(r, s) = (x_1 \pmod n, k^{-1}(z + rd_A))$, where z the leftmost bits of h of length $[1, n-1]$

2.3.3.2 ECDSA Signature Verification

The verifier, here party B, after receiving the signature pair (r, s) needs to verify that this is valid following the next steps:

1. Verify that r, s belong in the interval $[1, n-1]$.
2. Compute $e = hash(m)$ and $w = s^{-1} \pmod n$.
3. Compute $u_1 = ew$ and $u_2 = rw \pmod n$.
4. Compute $V = [u_1]G + [u_2]Q_A$. If $V = \mathcal{O}$, then reject the signature.
5. Convert the x-coordinate of V to an integer $\bar{x}$. If $u = \bar{x} \pmod n = r$, then accept the signature.

The Edwards-curve digital signature algorithm (EdDSA) as described in the original paper [BDL+12] is a signature scheme using Schnorr signatures and Twisted Edwards curves, aiming to speed up signature generation with the same security levels. The parameters of the scheme include an elliptic curve $\mathcal{E}$ over a finite field $\mathbb{F}_q$, where q is an odd prime, a base point $G \in \mathcal{E}(\mathbb{F}_{\mathrm{II}})$ of large prime order l and a collision resistant hash function $mathcalH$ with $2b$-bit output length, where $2^{b-1} > q$, so that elements of $\mathbb{F}_q$ and curve points in $\mathcal{E}(\mathbb{F}_q)$ can be represented by strings of b bits. The rational points of the group $mathcalE(\mathbb{F}_q)$ of $\mathbb{F}_q)$ have order $2^c l$, where 2^c is a public parameter and called the cofactor. For Ed25519 the curve $Curve25519$ and the hash function SHA-512 are used.

Each party participating in the EdDSA signature protocol possesses a key-pair, the private key d with $|d| = b$ chosen uniformly at random, and the public key $Q = [s]G$, with $|s| = b$ the least significant b-bits of $\mathcal{H}(d)$.

2.3.3.3 EdDSA Signature Generation

Let us assume that party A wants to send m signed to party B.

1. First, A selects randomly d and calculates $s = \mathcal{H}_{0,\ldots,b-1}(d)$ the b least significant bits of $\mathcal{H}(d)$.
2. A computes $r = \mathcal{H}(\mathcal{H}_{b,\ldots,2b-1}(d), m)$ and $R = [r]G$.
3. The signature is the tuple $(R, S) = ([r]G, r + \mathcal{H}(R, A, m) \pmod{l})$, where $R \in \mathcal{E}(\mathbb{F}_q)$ and $0 \leq S\ leql$.

2.3.3.4 EdDSA Signature Verification

The verifier, here party B, after receiving the signature pair (R, S) needs to verify that this is valid following the next steps:

1. Verify that R is indeed a point on the curve $mathcalE(\mathbb{F}_q)$ and S belongs in the interval $(0, l)$.
2. Compute $h = \mathcal{H}(R, Q, m)$, where Q is the public key of A.
3. If $2^cSG = 2^cR + 2^chQ$, then accept the signature.

Chapter 3

Side Channel Attacks on Elliptic Curves

> It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.
>
> *Sir Arthur Conan Doyle-Sherlok Holmes*

Cryptographic protocols used in a broad range of applications have one common property, their security relies on a hard mathematical problem. This problem is usually shown to be hard to solve in a polynomial time. In the case of ECC, this hard problem is ECDLP as described in Section 2.3.

The traditional security model assumes that the attacker has access to the public keys, he can request many signatures or messages to be encrypted and he knows how the protocol works. The only thing that the adversary does not know is the secret key, which he tries to retrieve by solving the underlying mathematical problems. In this setting, and depending on the security game in which we suppose the adversary participates, a cryptographic scheme can have different levels of security. For instance, in a perfectly secure scheme the ciphertext does not reveal any information about the plaintext, whether in a semantically secure scheme any information revealed on the ciphertext is not enough to extract useful information about the plaintext.

In practice, however, when a cryptographic algorithm is implemented on a specific platform, the same theoretical models are not relevant anymore. The classical security games for encryption or signatures do not recover keys, but reason about (in)distinguishability or forgeries. The ability to recover the secret key goes beyond that and amounts to a "total break". There is information leakage from the power consumption or electromagnetic emanation from the operating device, which can not be avoided and the use of countermeasures is necessary, in order to provide se-

curity. This sort of leakage, known as *side-channel* leakage, can be simulated by information theoretic models, which demonstrate the ability of the attacker to recover the keys. There are various *side-channel attacks* that make an implementation vulnerable and threatens its security; these attacks regarding ECC implementations are presented in this chapter. A systematic methodology to evaluate the leakage of a device based on several statistical tests, known as *Test Vector Leakage Assessment (TVLA)* is also explained in Section 3.6; it is used later for this thesis, in order to evaluate the leakage of proposed countermeasures. However, by using TVLA the leakage is only detected, it cannot be exploited. It is, therefore, used for an initial evaluation of a device and when leakage is detected, there are various attack methods to recover the secret keys.

3.1 Theoretical models for Side-Channel Attacks

Electronic devices that use cryptographic algorithms and store secret keys are vulnerable to various types of attacks. The non-invasive attacks are performed by observing the power or electromagnetic activity of devices, while they perform cryptographic operations, and without altering the circuits permanently. This sort of attacks are called *side-channel attacks*. There are passive attacks, where the attacker observes only the activity of the device while operating correctly, and active attacks, where the attacker intervenes and tries to induce faults during the execution of the cryptographic algorithms. In this thesis, we deal only with passive side-channel attacks.

Side-channel analysis as a method of extracting cryptographic keys was first presented by Kocher [Koc96], who noted that timing differences in the execution time of a modular exponentiation could be used to break instances of RSA [RSA78]. Subsequently, Kocher et al. [KJJ99] observed that the instantaneous power consumption could reveal information on intermediate states of any cryptographic algorithm, since the instantaneous power consumption has, in many cases, been shown to be proportional to the Hamming weight of the data being manipulated [BCO04]. It has also been shown that the electromagnetic emanations around a device can be exploited in the same way [GMO01,QS01]. Attacks based on these observations are typically referred to as Simple Power Analysis (SPA), Differential Power Analysis (DPA), Simple Electromagnetic Analysis (SEMA) and Differential Electromagnetic Analysis (DEMA).

During the design phase of a digital circuit, simulations of the power consumption are used, in order to determine if each component meets the design requirements or not. Power simulations describe the behavior of a device running algorithms with certain values. From an attacker's point of view, power simulations at a behavioral level are interesting, because they show data dependencies that may influence the security of the implementation.

The most widely adopted power models that are used to map simulated transitions to power traces are the Hamming-weight and Hamming-distance model. The Hamming weight of a value v is defined as the number of ones in its binary representation. Therefore, the Hamming-weight model is based on the number of ones in a certain value.

Definition 3.1.1. The *Hamming Distance* of two strings of the same length v_0, v_1 corresponds to the number of bit strings that differ from v_0 to v_1. This means that $HD(v_0, v_1)$ can be defined as follows: $HD(v_0, v_1) = HW(v_0 \oplus v_1)$.

The Hamming-distance model (HD) is used to deduce the power consumption values from the transitions that occur at the output of logic cells during dynamic power consumption of a circuit. Basically, by counting the $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions that occur in a digital circuit during a certain time interval, exploitable information over the manipulated secret key can be derived. According to [MOP07], in power simulations the Hamming-distance model assumes for simplicity that all cells contribute to the power consumption equally, and that there is no difference between $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions. However, in practice the power consumption of a $0 \rightarrow 1$ transition is higher than the one for $1 \rightarrow 0$ transition. The HD model is mostly used to describe the power consumption of buses and registers, and it is, therefore, appropriate to perform hardware side-channel attacks. Implicitly, since all software runs on hardware, this leakage model is also used for software attacks. The choice of the appropriate model depends on the measurement setup and the placement of the trigger.

The Hamming-weight (HW) model is simpler, in the sense that it takes into account the power consumption of data that are processed. The attacker assumes that the power consumption is proportional to the bits that are set to 1 in the value of interest. By observing the value of one bit before and after it is processed, we can notice differences in the power consumption. If the cell that processes the value v always stores the same value before processing v (it is for instance a register that is always initialized to 0), then the power consumption is directly or inversely proportional to the initial bits of this value. If the value that is stored in this initialization register is a random variable, then comparing the initial and processed values will not give any useful information to the attacker; the results will be independent. The HW model might seem simpler to simulate for a cryptographic devices, but it offers less side-channel information to the attacker and it is used when the HD model gives no useful information.

3.2 Simple Power Analysis on Elliptic Curves

Cryptographic implementations are vulnerable to simple power analysis (SPA) or simple electromagnetic attacks (SEMA) if there are distinct patterns in a trace,

which depend directly on the key. Kocher et al. [KJJ99] defined SPA in the following way:

Definition 3.2.1. Simple Power Analysis (SPA) is a technique that involves directly interpreting power consumption measurements collected during cryptographic operations. SPA can yield information about a device's operation as well as key material.

In practice, SPA is applied to a scenario when one or only a small number of power traces in available to the attacker. In the ECDSA setting for instance, the ephemeral key that is used in the protocol can be retrieved with SPA. Implementations that involve *conditional branching* are vulnerable to SPA. The scalar multiplication in ECDSA is implemented as a sequence of point doubling and point addition operations. If the implementation is done as in Algorithm 1, then SPA is applicable and an adversary is able to recover the secret scalar bits, as long as he can distinguish between doubling and addition patterns.

Coron [Cor99] was the first to observe that a straightforward implementation of scalar multiplication is vulnerable to SPA and he proposed countermeasures and regular algorithms to thwart this type of attack. Örs et al. [ÖOP03] demonstrated the first practical results of SPA on FPGAs for an ECC hardware implementation. The following figure, taken from [ÖOP03] shows in practice how the differences in the power consumption of doublings (0) and additions (1) can be identified and reveal the secret scalar.

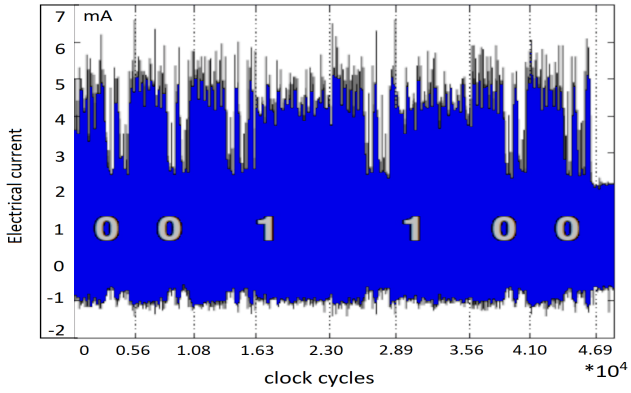

Figure (3.1) Power consumption trace of 160-bit double-and-add point multiplication [ÖOP03]

3.3 Differential Power Analysis on Elliptic Curves

Differential Power Analysis (DPA) attacks are widely used for attacking symmetric algorithms. A few published papers discuss DPA on ECC, mainly due to the fact that

ECC operations produce traces with millions of samples, which are hard to process in a DPA setting. Another reason why DPA is not broadly applicable to ECC, is the fact that session or ephemeral keys are often used, for instance in digital signatures. When the keys change regularly, then it is not possible to collect a large number of traces and exploit the dependencies between the keys and the power consumption.

DPA attacks need a large number of power traces that are collected while a cryptographic algorithm performs an interesting operation. The main advantage of DPA compared to SPA is the limited knowledge on the cryptographic device; usually only knowledge of the cryptographic algorithm used, is adequate to perform a DPA attack. DPA reveals the data dependencies of the power consumption; it usually requires sophisticated statistical analysis of the power traces.

The general attack strategy described in [MOP07] gives a good insight into DPA. For the case of scalar multiplication it consists of the following steps:

1. Choose an intermediate result of the executed algorithm. In the case of a DPA against a scalar multiplication, an interesting intermediate result is the output of each round of the iteration over the scalar bits.

2. Measure Power Consumption. Traces t_i of n samples can be collected each time the interesting intermediate result is calculated. These traces need to be aligned using several alignment techniques from signal processing.

3. Calculating Hypothetical Intermediate Values. For ECC scalar multiplication, these hypothetical values can be the multiples of the base point $m \cdot \boldsymbol{P}, m \in \mathbb{Z}$.

4. Mapping Intermediate Values to Power Consumption Values. Power consumption values are calculated based on simulations using the Hamming-weight or the Hamming-distance models. The quality of the simulated traces is based on the attacker's knowledge of the device and it will affect the success rate of the DPA attack.

5. Comparing the Hypothetical Values with Power Traces. At this point, the attacker compares the hypothetical values to the real power traces, i.e. simulated traces about all possible multiples of the base point, are compared to the real traces. Those traces that give the highest correlation to the real traces are the ones which are produced by the correct multiple $m_j \cdot \boldsymbol{P}$.

Similar attack strategies can be applied when we want to attack other operations in ECC, for instance finite field operations, modular reductions etc.

There are various statistical metrics, such as correlation coefficient, Pearson's coefficient, difference of means, that can be used to find the highest values for this comparison. Usually, distinguishers that take into account variances, for instance distance of means, are the ones that would give the highest matching results.

Coron [Cor99] describes an interesting DPA attack on scalar multiplication. Assuming that we know how the points are represented in memory during computation, we can correlate bits of the point that is processed with the specific bit of the point stored in memory and in this way compute if the processed point is already used by the card or not.

As we mentioned in the beginning of this section, DPA on ECC is quite demanding in terms of processing power and memory requirements. A DPA-like attack on pairings is described in [PV04]. More precisely, the secret key, represented by a point $S_{ID} = (x_1, y_1)$, is recovered by guessing the value y_1 bit-by-bit and comparing it to the decryption signal taken during proper executions of the algorithm. In CT-RSA 2018 an attack by Samwel et al. [SBB$^+$18] on EdDSA is another example of a practical DPA attack on an ECC protocol. The deterministic part of EdDSA is exploited, by performing DPA on the underlying hash function, SHA-512. This indicates that a protocol using ECC can be attacked in practice in various ways by attacking other building blocks of the protocol than the ECC operations. However, a practical DPA attack on a real-world ECC implementation targeting purely elliptic curve operations is still not published.

3.4 Collision Attacks

The definition of collision attacks in [MOP07] is quite simple and complete.

Definition 3.4.1. Suppose that we are able to observe the power consumption of two consecutive executions of an encryption algorithm with two different inputs d_i and d_i^*, and the unknown key k_j. In a collision attack we exploit the fact that an intermediate value v_{i_j} can be the same in these two different execution, that means $v_{i,j} = f(d_i, k_j) = f(d_i^*, k_j)$.

The collision of the two intermediate values can occur only for certain key values, which allows reducing the key space for the secret key. There are various types of attacks that are based on this observation, and this is what we are going to describe in this section.

3.4.1 Collision-correlation Attacks

As mentioned above, collision attacks exploit leakages by comparing two portions of the same or different traces exploiting the same power being consumed when values are reused. The Big Mac attack [Wal01] is the first theoretical attack on public key cryptosystems, in which only a single trace is required to observe key dependencies and collisions during an RSA exponentiation. Witteman et al. performed a similar attack on the RSA modular exponentiation even in the presence of blinded messages [WvWM11]. Clavier et al. introduced horizontal correlation power analysis (CPA), as a type of attack where a single power trace is enough to recover the

private key [CFG+10]. They also extended the Big Mac attack by using different distinguishers, such as the correlation factor between Hamming weights. Compared to the Big Mac attack, where the attacker is able to distinguish between operations (squarings and multiplications), by horizontal collision-correlation it is possible to validate guesses based on the manipulation of intermediate results. Therefore, this attack is also applicable when the implementation is regular, which is not the case for the Big Mac Attack.

A special type of collision attack is the *doubling attack* proposed by Fouque and Valette [FV03]. The main assumption of this attack is that an adversary can distinguish collisions of power trace segments (within a single or more power traces) when the device under attack performs the same computation twice with the same data, even if the adversary is not able to tell which exact computation is done. Collision of two computations will not reveal the value of the operand. Yen et al. extended this attack to the *Refined Doubling Attack (RDA)* [YKMH05], where the adversary is assumed to be able to detect the collision between two modular squarings, i.e. detecting if the squared value is the same or not. Collisions of computations cannot be distinguished; the only knowledge obtained is that $k_i = k_{i-1}$ if a collision is detected. Based on the derived relationship between every two adjacent private key bits (either $k_i = k_{i-1}$ or $k_i \neq k_{i-1}$) and a given bit (e.g., k_0 or k_{m-1}), all other private key bits can be derived uniquely. RDA is a powerful attack technique that works against some scalar multiplication algorithms, which are resistant against the doubling attack (e.g. the Montgomery power ladder).

3.4.2 Horizontal Attacks

An interesting class of side-channel attacks is the *Horizontal Analysis* attack, where a single trace is used to recover the secret scalar. The main characteristic of the traces that makes horizontal attacks possible lies in the fact that the operation sequences of doubling-adding and doubling-doubling can be distinguished. The attacker applies the classical correlation analysis using different parts of time samples in the same side-channel trace to recover the secret scalar bit-by-bit. This technique can be useful to attack protected implementations, where the secret value or unknown input is blinded. The first horizontal attacks were applied to RSA implementations; extension of those to ECC implementations is straightforward, since scalar multiplication and exponentiation algorithms have the same operation steps.

The so-called *Big Mac* attack from Walter [Wal01] is the first attack of this kind, where squarings (**S**) are distinguished from random products or multiplications by a constant (denoted both as **M** for the sake of simplicity) and the secret exponent of an RSA exponentiation can be recovered from a single execution curve. The distinction is possible because of the different amount of gate switching activity in (the same) underlying multiplier. In the simulated traces from the original paper, the difference in the statistical values corresponding to **M** or **S** is quite large; it increases

as the modulus length gets larger with 0% error rate for modulus larger than 768 bits. Although the principle of Big Mac could be applied to ECC implementations, where doublings and additions are distinguishable operations, a practical direct application of the attack on real measurements, performed by Danger et. al [DGH$^+$16] failed, mainly due to the small scalar length compared to the equivalent exponent length in RSA. The authors of [DGH$^+$16] proposed an improvement of Big Mac, where combining several patterns depending on scalar bits, made it possible to recover the scalar.

The term *horizontal* was first introduced by Clavier et al. in [CFG$^+$10], where the authors performed a horizontal correlation analysis to compute the correlation factor on several segments extracted from a single execution curve of a known message RSA encryption. More specifically, their proposed method starts by finding a sequence of elementary calculations $(C_i)_j$ that processes the same mathematical operation (e.g. field multiplication) and depends on the same part of the secret scalar. The outputs O_{i_j} of the calculations $C_i(X_i)$ that depend on the same input value X_i will give high correlation results and in this way, they can be distinguished from outputs of computations with different input values. Horizontal correlation analysis was performed on RSA using the Pearson correlation coefficient in [CFG$^+$10] and triangular trace analysis of the exponent in [CFG$^+$12].

The first horizontal technique relevant to ECC is the doubling attack, presented by Fouque and Valette in [FV03]. The most recent attack, proposed by Bauer et al. in [BJPW14], is a type of *horizontal collision correlation* attack on ECC, which combines atomicity and randomization techniques. A basic assumption of collision attacks is the fact that an adversary is able to distinguish when two field multiplications have at least one common operand. Based on this assumption, their attack consists of the following steps:

- Identify two elementary calculations in the field C_1, C_2 that are processed N times with inputs from the same distribution. The correlation between the random output values O_1, O_2 must depend on the same secret sub-part s.
- For each of the N processings of C_i get an observation l_j^i, with $j \in [1, ..., N]$.
- Compute the Pearson correlation coefficient on the two samples of observations $\rho = \rho((l_j^1)_j, (l_j^2)_j)$.
- Deduce information on the secret scalar from ρ by using an appropriate distinguisher that shows which observation is more similar to the real secret value.

Simulated traces show that the horizontal collision correlation attack is applicable to atomic implementations 2.2.5 and to implementations based on unified addition formulæ over Edwards curves.

Two publications on blinded asymmetric algorithms propose the combination of horizontal and vertical techniques, in an attempt to provide more practical attacks against blinded implementations and avoid the complex signal processing phase. Bauer et al. [BJPW13] at Indocrypt 2013, presented an attack on RSA blinded exponentiation based on this approach. They took advantage of the side-channel leakage of the entire long-integer modular multiplication without splitting the trace into parts of single precision multiplications. However, their attack requires a small public exponent (no greater than $2^{16} + 1$) and an exponent blinding factor smaller than 32 bits. Their observation that the scalar blinding does not mask a large part of the secret value, led Feix et al. [FRV14] a year later to exploit this vulnerability vertically on a ECC implementation. The most significant part of the blinded scalar can be recovered with a horizontal attack. The least significant part of the scalar is retrieved by using vertical analysis (several execution traces) and the information leaked in the previous steps of the attack.

3.4.3 Vertical Attacks

A recent classification of attacks has categorized all the statistical attacks on multiple traces as Vertical Analysis, in comparison to the Horizontal Analysis that is performed on one or a few traces. Indeed, these techniques combine a single time sample t on many side-channel traces to perform the analysis leading to the recovery of the secret data manipulated at this instant t in a “vertical" way.

The term *vertical* is first mentioned in the paper of Clavier et al. [CFG$^+$10], to categorize the techniques that require many traces, in order to extract a key. It can be that many points on the same trace are used or the same point in different execution instances; in both cases at least two power execution traces are required and they should be manipulated together by the attacker. The classical DPA and CPA techniques thus fall into this category. In Indocrypt 2014 Feix et al. [FRV14] referred to the vertical collision correlation, when the attacker uses the fact that the scalar blinding does not mask a large part of the secret. This side-channel vulnerability can be exploited vertically, i.e. using several execution traces of the potential values for each input bit, and the attacker can recover this unmasked part with collision correlation. Moreover, the least significant part of the secret can be recovered using vertical collisions. By guessing the next w unknown bits of scalar k, we can compute guessed blinded scalars $d^{(i)}$. Then a classical vertical correlation attack can be performed to validate the guesses.

Vertical leakage was also exploited in [DPN$^+$16], where the authors used multiple template traces to perform pattern matching, when the propagation of carry was during scalar multiplication (and therefore, no visual horizontal difference in the traces). The exploitable vertical leakage comes from the HW of the value stored in the register or the HD between two values stored in the same register, which is basically the same principle as in DPA types of side-channel attacks.

3.5 Template Attacks

The most powerful SCA attack from an information theoretic point of view is considered to be a template attack (TA).

3.5.1 Theoretical Aspects of Template Attacks

In the original paper by Chari et al. in [CRR02], template attacks are introduced as a combination of statistical modeling and power analysis attacks consisting of two phases, as follows:

- The first phase is the *profiling* or *template-building* phase, where the adversary builds templates to characterize the device by executing a sequence of instructions on fixed data. Focusing on an "interesting pattern" or finding the points of interest is very common in this phase.
- The second phase is the *template-matching phase*, in which the adversary matches or correlates the templates to actual traces of the device. By applying some signal processing and classification algorithms to the templates, it is possible to find the best matching for the traces.

In this type of attacks, the adversary is assumed to have in his possession a device which behaves similarly to the device under attack (target device), in order to build template traces. In his device he can simulate the same algorithms and implementations that run in the target device. For the template-matching phase several distinguishers and classification algorithms are proposed; in the next section the most common classifiers are presented.

The practical application of TAs is shown on several cryptographic implementations such as RC4 in [RO04] and elliptic curves in [MBO+05]. Medwed and Oswald demonstrated in [MO09] a practical template attack on ECDSA. Their attack required an offline DPA attack on the EC scalar multiplication operation during the template-building phase, in order to select the points of interest. They also need 33 template traces per key bit. Attacks against ECDSA and other elliptic curve signature algorithms only need to recover a few bits of the ephemeral scalar for multiple scalar multiplications with different ephemeral scalars and can then employ lattice techniques to recover the long-term secret key [RS01, MHMP13, BvdPSY14]. The lattice attack works by constructing a lattice problem from the obtained digital signatures and side channel information, and then applying lattice reduction techniques to solve this problem. Nguyen and Shparlinski showed how to recover the secret key of 160-bit ECDSA using only seven consecutive bits that leaked from signatures with ephemeral keys [NS03]. However, if an attacker only gets a single trace, he still needs to recover sufficiently many bits of an ephemeral scalar from side-channel information to be able to compute the long term key.

3.5.2 Common Distinguishers

In this section, the most common distinguishers used in SCA for correlation analysis and template-matching are presented. Machine Learning techniques for classification and clustering are broadly used in SCA, in order to distinguish between traces with high noise ratios.

According to [HIM+13], unsupervised clustering is generally useful in side-channel analysis when profiling information is not available and an exhaustive partitioning is computationally infeasible. The authors presented an attack on an FPGA-based elliptic curve scalar multiplication using the k-means method. In [PITM14], Perin et al. used unsupervised learning to attack randomized exponentiations.

Lerman et al. showed in [LPB+15] that Machine Learning techniques give better classification results when there is limited ability of the adversary to perform profiling of the device and in a high dimensionality context, where many parameters affect the leakage of the device. Indeed, combining three side-channel leakages and a clustering-based approach for non-profiled attacks, gives higher success rates than traditional template attacks, as shown by Specht et al. in [SHKS15].

The success rate of Online Template Attacks (presented in the next section) are significantly improved in [OPB16] by using the k-nearest neighbour approach, naïve Bayes classification and the Support Vector Machine method for template classification. In order to explain these techniques, we first give the definition of the Euclidean distance and the Pearson correlation coefficient.

3.5.2.1 Euclidean Distance

The Euclidean distance between two points is defined as the square root of the sum of the squares of the differences between the corresponding point values:

$$d_{EUC} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}\,.$$

In the SCA setting, a realization of a random variable X corresponds to x. A sample of n observations or traces of X is denoted by $(x_i)_{1\leq i\leq n}$, where the index i denotes the different observations or the different time when an observation occurs in the same trace.

3.5.2.2 Pearson correlation

The Pearson correlation coefficient measures the linear independence between two observations X and Y:

$$\rho(X,Y) = \frac{n\sum_i x_i y_i - \sum_i x_i \sum_i y_i}{\sqrt{n\sum_i x_i^2 - (\sum_i x_i)^2}\sqrt{n\sum_i y_i^2 - (\sum_i y_i)^2}}$$

In SCA, the Pearson correlation coefficient is used to describe the difference in the Hamming weight between the observations from the device under attack and the key hypothesis.

3.5.2.3 Mutual Information

The Mutual Information distinguisher uses the value of the Mutual Information between the observed measurements and a hypothetical leakage to rank key guesses. From a theoretical point of view, Mutual Information Analysis is a non-profiled metric that can detect any kind of data dependency in the physical measurements. The mutual information between two random variables **X** and **Y** can be calculated as follows:

$$I(X;Y) = H(X) - H(X \mid Y) = H(X) + H(Y) - H(X,Y) \tag{3.1}$$

where $H(X) = \sum_{x\in\mathcal{X}} \Pr[X = x] \cdot \log(\Pr[X = x])$ the Shannon entropy of a random variable **X** on a discrete space $\mathcal{X}$. The conditional entropy $H(X \mid Y) = -\sum_{yinY} \Pr(X = x \mid Y = y) \log_2 \Pr(X = x \mid Y = y)$ quantifies the amount of information needed to describe the outcome of a random variable X given the value of a known random variable Y.

3.5.3 Classification Algorithms

3.5.3.1 Naïve Bayes Classification

The naïve Bayes classification method is based on probability concepts, and more precisely on Bayes theorem for conditional probabilities of independent events [Bra13]. According to the conditional probability model, let $\mathbf{x} = (x_1, \ldots, x_n)$ be the vector of problem instances (independent variables) to be classified, each one having a feature n. Each instance is assigned a probability $p(c_j|x_1, \ldots, x_n)$, for $1 \leq j \leq k$ and k possible classes. The set of classes $c_1, c_2, \ldots, c_k$ is mutually exclusive and exhaustive. Using Bayes' theorem, the posterior conditional probability is $p(c_j|\mathbf{x}) = \frac{p(c_j)\, p(\mathbf{x}|c_j)}{p(\mathbf{x})}$ for $1 \leq j \leq k$. Assuming that each event and posterior probability is independent on each previous event, the conditional distribution over the class variable c is $p(c_j|x_1, \ldots, x_n) = \frac{1}{Z} p(c_j) \prod_{i=1}^{n} p(x_i|c_j)$ for $1 \leq j \leq k$ where the evidence $Z = p(\mathbf{x})$ is a scaling factor dependent only on $x_1, \ldots, x_n$, that is, a constant if the values of the feature variables are known.

The naïve Bayes classifier is a function that combines the naïve Bayes probability model with a decision rule. One common rule is to pick the hypothesis that is most probable; that is the maximum value of the a posteriori probability. For each

class c_i, we pick as classifier the class index that gives the maximum value for an event.

3.5.3.2 k-Nearest Neighbour

The k-Nearest Neighbour Classification *kNN* is a classification method based on the closest instances of the training set to the unlabeled data. Basically, according to [Bra13], it consists of the following two steps:

1. Choose the number of k closest instances (from the training set) to the sample.
2. The majority label (class) for the chosen closest instances will be class for the unlabeled data.

The distance metric plays an important role as we determine the closest instance. In *kNN*, we can use the Euclidean distance or the Manhattan distance. The value k indicates the number of the already-classified closest instances that are chosen in order to classify the next unlabeled data. The default value is 1, but with larger value for k it is possible to obtain higher success rate with less template traces. Figure 3.2 shows an example with $k = 2$ and $k = 4$ close instances; if the new sample is closer to A, it will be classified in the “A-class".

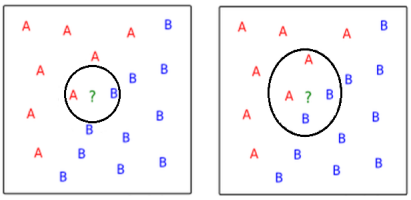

Figure (3.2) kNN method for $k = 2$ and $k = 4$

3.5.3.3 Support Vector Machine

A Support Vector Machine (SVM) is a supervised learning model that produces a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new examples. Figure 3.3 shows such a hyperplane. An optimal hyperplane, as defined in [Alp10], is the one that gives the largest minimum distance to the training points, because in this way noisy data will still be classified correctly. Therefore, the optimal separating hyperplane maximizes the margin of the training data. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted

to belong to a category based on which side of the gap they fall on. The classifier in an SVM can be non-linear for data sets that are not easily separable, but in our analysis a linear classifier gives very good results. In machine learning, support vector

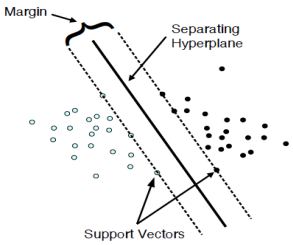

Figure (3.3) SVM: distance to the hyperplane for two sets of training data

machines (SVMs, also support vector networks[1]) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

3.6 Test Vector Leakage Assessment

Test Vector Leakage Assessment (TVLA) is a leakage detection procedure, initially proposed by Cryptography Research (CRI) [GJJR11] and used as a first step towards evaluation of the SCA resistance of device under test. A generic univariate test is used to scan the traces obtained from the device, which is evaluated as non-leaky if the tests at all points of every trace are below a certain threshold. The statistical tests are chosen in a way that is independent of the leakage model.

More precisely, the case that there is no leakage (null hypothesis) is tested against the case that there is leakage at a certain intermediate point L_P. Let n_{tr} be the number of traces that the evaluator collects and n_s the number of samples in each trace. Following the notation of [ZDD$^+$17], let $\mathbf{L} = \{L_1, \ldots, L_{n_s}\}$ be the measurement traces representing the realization of our implementation with mean values $\bar{L}_i$ and the noise follows the normal distribution $\mathcal{N}_{(0,1)}$. The null hypothesis for the traces means that the expected value from our measurements is the same as the measured value, if there is no leakage, i.e. $\bar{L}_{exp} = \bar{L}_i \quad \forall i \in \{1, \ldots, n_s\}$.

Following the widely adopted methodology for t-test from [GJJR11, NLD15, TG16, CMV$^+$17], there should be two sets of traces, A and B, with $n_{tr}/2$ traces in each set; half of the traces are taken with a fixed input and half with random

input. If the null hypothesis holds, there should be no differences in the t-test statistics measured from each trace set. The Welch's t-test is commonly used for TVLA evaluations. The test statistic value is:

$$s_i = \frac{\bar{L}_{i,A} - \bar{L}_{i,B}}{\sqrt{\frac{\sigma^2_{i,A}}{n_A} + \frac{\sigma^2_{i,B}}{n_B}}}. \tag{3.2}$$

There is a broad bibliography on symmetric key cryptographic evaluations using TVLA; theoretical advances of the technique, multivariate higher order tests [MOBW13, SM16, DCE16] and various statistical metrics are presented [BGN+14, BGG+14, PV17]. However, since its introduction to public-key algorithms in 2011 [JRW11], it was rarely used for theoretical analysis and practical evaluation in this research field. Most notably, Tunstall and Goodwill [TG16] give an overview of cases and algorithms that can be applied during public key TVLA evaluations. Nascimento et al. [NLD15], Chmielewski et al. [CMV+17] used it for evaluation of Curve25519 on Chipwhisperer and the complete Weierstrass formulæ on an FPGA respectively.

For the Welch's t-test, current TVLA evaluations use the critical value of 4.5 [GJJR11, NLD15, TG16, CMV+17], which corresponds to a statistical significance level of $\alpha < 0.00001$ for the univariate test. The threshold of 4.5 is sufficient when symmetric-key algorithms are evaluated, because of the small number of samples in each trace (order of hundreds). However, in public-key cryptography, the traces are usually larger, requiring thousands or million samples for capturing a meaningful part of the operation under evaluation. In [CMV+17] the authors observe some peaks above the threshold for the protected implementation, which ended up to be ghost peaks. Their rationale is that first they observed few of these peaks in the fixed versus random t-test. Then, they took two trace sets with random values and applied the same test. The peaks observed before did not appear again at the same spots, meaning that they were not dependent on the input.

In most published works, the significance level of $a < 0.00001$ does not consider the total number of samples on the trace. As noted in [ZDD+17], the overall significance level increases as the number of leakage points on the trace increases. Therefore, the authors propose to adjust the significance level according to the number of points on a trace. For long traces, meaning for more than 10^5samples per trace, the overall test statistic value will be larger than 4.5 and therefore a non-leaky device can not pass the TVLA t-testing with the critical value of 4.5. Hence, Balasch et al. [BGG+14] suggested raising the critical value to 5 for longer traces based on numerical experiments. For longer traces, as the ones obtained from the implementation of public key algorithms, a non-leaky device can not pass the t-test even with this higher value of 5.

In general, t-tests are effective in detecting some vulnerabilities of an implementation, but they do not give specific results on the type of leakage. TVLA evalua-

tions are hardly ever applied correctly. All detected vulnerabilities should be verified with repeating the tests on a second acquisition, which is independently generated. However, since trace acquisitions are expensive, there are other alternatives proposed, such as the correlation-based method [DS16], which can give better results with fewer traces, because larger parts of the trace are exploited and more efficient statistics is used. Moreover, TVLA evaluations usually ignore false negative type of errors (type II error rate β), which indicate the probability to reject a false null hypothesis. The statistical power of a test is defined as the probability of correctly rejecting a false null hypothesis [MOBW13] $\pi = 1 - \beta$. TVLA, as it is usually performed, does not consider the power of the statistical procedure it proposes. Mather et al. [MOBW13] propose a thorough analysis to check the impact on statistical power of the tests, in order to estimate the minimum sample size required to detect an effect of a given size. Higher statistical power indicates increased robustness and minimizes the requirements for large sample sets; avoiding expensive and time-consuming acquisitions can speed-up the evaluation process. This is explained concretely in a tutorial [Pro18], but it is out of scope of this thesis.

3.6.1 TVLA for public key

One of the applications of Welch's t-test is to test the location of one sequence of independent and identically distributed random variables, as the ones obtained from SCA measurements. In such a trace set, we are interested in s different null hypotheses, $H_1, \ldots, H_s$, where s is the number of samples in each trace, and we would like to check if all of them are true. The probability of making one (or more) false discovery when performing multiple hypotheses tests, the so-called family-wise error rate (FWER) or type I errors, is related to the number of samples. The more samples we get per trace, the higher the FWER will be. In order to calibrate the significance level, and consequently the threshold of the test, when multiple sequences of variables are tested, the Šidák correction should be applied.

The above mentioned calibration makes more sense in the case where we deal with public key algorithms, where the number of samples per trace are in the order of millions. The Šidák correction as defined in [FHY07] is

$$a_{SID} = 1 - (1 - a)^{n_s}. \tag{3.3}$$

This formula appeared recently in the side-channel setting in [ZDD+17], where the authors identified the need to correlate the significance level with the total number of univariate tests. More precisely, they showed that for a trace set of 10^6 samples per trace, the probability that the t-test will fail for the ± 4.5 threshold is 0.9987, reducing only to the half for a threshold value of ± 5.

We use this formula to create a Matlab script for evaluating long traces obtained from a public key implementation in Chapter 6, in order to obtain more accurate t-test results for our implementation.

Chapter 4

Online Template Attacks

> You show the world as a complete, unbroken chain, an eternal chain, linked together by cause and effect.
>
> *Hermann Hesse, Siddhartha*

In this chapter, we present *Online Template Attacks* (OTA), a novel attack technique that resides between horizontal and template attacks. While the terminology *template* is used, OTA is not a typical template attack; i.e. no preprocessing template-building phase is necessary. The templates of certain operations are compared with parts of the target trace and the secret key can be recovered bit-by-bit. The characterization *online* comes from the fact that templates are created *after* the acquisition of the target trace.

4.1 Introduction

Side-channel attacks exploit various physical leakages of secret information or instructions from cryptographic devices and they constitute a constant threat for cryptographic implementations. We focus here on power-analysis and electromagnetic attacks that exploit the power-consumption and electromagnetic-emanation leakage from a device running some cryptographic algorithm. Attacking elliptic curve cryptosystems (ECC) with natural protection against simple side-channel attacks, e.g. implementations using complete Weierstrass formulas [RCB16] or Edwards curves, is quite challenging. The Edwards curves, for instance, proposed by Edwards in 2007 [Edw07] and promoted for cryptographic applications by Bernstein and Lange [BBJ+08], can provide fast and complete formulas for addition and doubling. This is why these types of curves are appealing for memory-constrained devices and at the same time resistant to classical simple power analysis (SPA) techniques. Although considered a very serious threat against ECC implementations, differential

power analysis (DPA), as proposed in [Cor99, KJJ99], cannot be applied directly to ECDSA or ephemeral Diffie-Hellman because the secret scalar is used only once. This is incompatible with the requirement of DPA to see large number of power traces of computations on the same secret data. In order to attack various asymmetric cryptosystems, new techniques that reside between SPA and DPA were developed; most notably collision [Wal01, SWP03, FV03, YKMH05, HMA+08, BJPW14] and template attacks [RS01,MO09,MHMP13]. The efficiency of most of those collision-based attacks is shown only on simulated traces; no practical experiments on real ECC implementations have verified these results. To the best of our knowledge, only two practical collision-based attacks on scalar multiplication algorithms have been published, each of which relies on very specific assumptions and deals with very special cases. Hanley et al. exploit collisions between input and output operations of the same trace [HKT15]. Wenger et al. in [WKK13] performed a hardware-specific attack on consecutive rounds of a Montgomery ladder implementation. However, both attacks are very restrictive in terms of applicability to various ECC implementations as they imply some special implementation options, such as the use of López-Dahab coordinates, where field multiplications use the same key-dependent coordinate as input to two consecutive rounds. In contrast, our attack is much more generic as it applies to arbitrary choices of curves and coordinates, and many scalar multiplication algorithms.

4.1.1 Related Work

In Chapter 3, we presented in detail collision attacks against public key algorithms that use a single or a few traces to exploit data dependencies during execution of the same operations. Starting with the Big Mac attack [Wal01] by Walter to the doubling attack [FV03] by Fouque and Valette and the more recent attacks on ECC using horizontal collision-correlation [HKT15], we observe that the trend is moving towards horizontal type of attacks. The main goal of evaluators is to discover the secret key with as few leakage traces as possible.

As presented in Section 3.5, template attacks are a combination of statistical modeling and power-analysis attacks consisting of two phases, as follows. The *profiling* or *template-building* phase, where the attacker builds templates to characterize the device by executing a sequence of instructions on fixed data. The second phase is the *matching phase*, in which the attacker matches the templates to actual traces of the device. It is usually enough to recover a few bits of ephemeral keys during ECDSA algorithms and the rest of the bits are recovered by other techniques, such as lattice-based attacks. In a nice overview paper on lattice-based attacks, Wong showed how to recover 156-bit nonces used in ECDSA signatures with 7 known bits [Won15]. When only 6 bits are known, the lattice-base reduction technique LLL [LLL82] needs a minimum of 78 tuples (signatures and truncated hashes) to recover the nonces.

Still, very few practical attacks are published on ECC implementations. For instance, Medwed and Oswald demonstrated in [MO09] a practical template attack on ECDSA. Their attack required an offline DPA attack on the scalar multiplication during the template-building phase and 33 template traces per key bit. Another template attack on ECC is presented in [HIM+13]. This attack exploits register location based leakage using a high-resolution inductive EM probe; therefore, the attack is considerably expensive to execute. A template attack on a windowed-NAF ECC algorithm is presented in [ZWMZ14]. However, this attack is applied to an implementation that is not protected with either, scalar randomization or base-point randomization. Furthermore, contrary to our approach, all of the above attacks require multiple traces to construct a template.

4.1.2 Our Contribution

In this chapter we introduce an adaptive template-attack technique, which we call *Online Template Attacks* (OTA). With this technique the attacker is able to recover a complete scalar from only one power trace of a scalar multiplication using this scalar. The attack is characterized as *online*, because we create the templates *after* the acquisition of the target trace. While we use the same terminology, our attack is not a typical template attack; i.e. no preprocessing template-building phase is necessary. Our attack functions by acquiring one target trace from the device under attack and comparing patterns of certain operations from this trace with templates obtained from the attacker's device that runs the same implementation. Pattern matching is performed at suitable points in the algorithm, where key bit related assignments take place by using an automated module based on the Pearson correlation coefficient.

The attacker needs only very limited control over the device used to generate the online template traces. The main assumption is that the attacker can choose the input point to a scalar multiplication, an assumption that trivially holds even without any modification to the template device in the context of ephemeral Diffie-Hellman. It also holds in the context of ECDSA, if the attacker can modify the implementation on the template device or can modify internal values of the computation. This is no different than for previous template attacks against ECDSA.

Our methodology offers a generic attack framework, which is applicable to various forms of curves (Weierstrass, Edwards and Montgomery curves) and implementations. As a proof of concept, we attack the doubling operation in the double-and-add-always algorithm. Contrary to the doubling attack [FV03], our attack can be launched against right-to-left algorithms and Montgomery ladder. We further note that Medwed and Oswald perform a very special template attack based on a set of assumptions: DPA performed in advance to find intermediate points for templates, 33 template traces, implementation with Hamming-weight leakage that is hardware dependent and applicability only to ECDSA. Online template attacks do not have these restrictions, they need only a single target trace, and only a single template

trace per key bit. The advantages of our attack over previously proposed attacks are the following:

- It does not require any cumbersome preprocessing template-building phase, but a rather simple post-processing phase.
- It does not assume any previous knowledge of the leakage model.
- It does not require full control of the device under attack.
- It works against SPA-protected and some DPA-protected implementations with unified formulas for addition and doubling.
- Countermeasures such as scalar randomization and changing point representation from affine to (deterministic) projective representation inside the implementation do not prevent our attack.
- It is applicable to the Montgomery power ladder and to constant-time (left-to-right and right-to-left) scalar multiplication algorithms.
- It is experimentally confirmed on an implementation of double-and-add-always scalar multiplication on the twisted Edwards curve used in the Ed25519 signature scheme.

We note here that our attack is a chosen input attack, meaning that the adversary needs to control the input of a scalar multiplication (but not the scalar). Most ECC implementations use inputs in affine (or compressed affine) coordinates and internally convert to projective representation. In this chapter we show how to apply the attack if an attacker controls either the projective coordinates input, or the affine input (even if it is compressed). Online template attacks require only one *target trace* and one *online template trace* per key bit. We can, therefore, claim that our technique demonstrates the most efficient practical side-channel attack applicable to ephemeral-scalar ECC. When applied to ECDSA, the proposed attack can be used in combination with lattice techniques similar to [RS01, BvdPSY14], in order to derive the whole private key from a few bits of multiple ephemeral keys.

The fact that OTA is a chosen input attack is the main difference between our technique and the extend-and-prune technique of the original paper about template attacks [CRR02] featured on RC4. The extend-and-prune technique is an iterative process, where at each step one more segment of the sample trace is unrolled, which uses more bits of the unknown key. Each extension results in several hypotheses about the operation being performed. Templates are used to prune the possible hypothesis values on the key bits while controlling the error probability to be under a certain threshold. The extended version of OTA, including error detection and correction and presented in Section 4.5.5, is very similar to the pruning process. If the matching percentage of $2\boldsymbol{P}$ and $3\boldsymbol{P}$ with the target trace are close enough to draw

a conclusion, then the template traces of $4\boldsymbol{P}$, $5\boldsymbol{P}$, $6\boldsymbol{P}$ and $7\boldsymbol{P}$ can be used; with 4 template traces for the 3rd most significant bit, we can infer both the 3rd and the 2nd most significant bits. This process is quite simple compared to the pruning process described in [CRR02].

Having presented the contribution of OTA in the side-channel world, it is desirable to clarify the contribution of the author in this context. In the original publication [BCP+14, BCP+17], the author established the theoretical primitives that would make the attack applicable to many scalar multiplication algorithms and developed the attack technique on which the practical experiments are based. After the successful application of the power analysis OTA on the Edwards curve 25519, she investigated the wide applicability of the attack to other platforms and curves, resulting in the work of "Dismantling real-world ECC with Horizontal and Vertical Template Attacks" [DPN+16]. In the follow-up work from Özgen et al. [OPB16], the author proposed and co-supervised a Master thesis project on using distinguishers from machine learning (classification methods) together with OTA, which resulted in correct predictions on the scalar bit and a successful application of OTA.

4.1.3 Organization of this Chapter

This chapter is organized as follows. We introduce and explain the theoretical aspects of OTA in Section 4.2. Section 4.3 gives specific examples of how the attack applies to different scalar multiplication algorithms. Section 4.4 presents the practical OTA on double-and-add-always scalar multiplication on an ATMega card. In Section 4.5 the attack on Weierstrass curves on a Cortex-M4 micro-controller is presented in detail together with the proposed error detection and correction technique. Distinguishers from the machine learning field are presented in Section 4.6 and they are applied during the template matching phase of OTA. A discussion on vulnerabilities of implementations and efficient countermeasures against OTA concludes the chapter in Section 4.7.

4.2 Online Template Attacks

This section presents the theoretical aspects of online template attacks as applied to scalar multiplication algorithms.

4.2.1 Defining Online Template Attacks

We define an online template attack (OTA) as a side-channel attack with the following conditions:

1. The attacker obtains only one power trace of the cryptographic algorithm involving the targeted secret data. This trace is called the *target trace*. The device, from which the target trace is obtained, is the *target device*. The fact that only one target trace is necessary for the attack, makes it possible to attack scalar multiplication algorithms with an ephemeral, randomized or blinded scalar.

2. The attacker is generating template traces *after* having obtained the target trace. These traces are called *(online) template traces*.

3. The attacker obtains the template traces on the target device or a similar device, [1] *with very limited control over it* i.e. access to the device to run several executions with chosen public inputs. The attacker does not rely on the assumption that the secret data are the same for all template traces.

4. At least one assignment in the scalar multiplication algorithm is made depending on the value of particular scalar bit(s), but there are no branches with key-dependent computations. Since we are attacking the doubling operation, this key-dependent assignment should be during doubling. As a counterexample, we note that the binary right-to-left add-always algorithm for Lucas recurrences [Joy07] is resistant to the proposed attack, because the result of the doubling is stored in a non-key-dependent variable.

In the following we show that online template attacks are feasible and can be applied against implementations of various scalar multiplication algorithms. In fact, we show that we need only a single template trace per scalar bit. Transfer of the approach to the corresponding exponentiation algorithms (for example in RSA or DSA) is straight-forward. Transfer to other cryptographic algorithms is clearly not trivial; we consider online template attacks as a specialized means to attack scalar multiplication and exponentiation algorithms.

We present the theoretic primitives of the attack and verify our theory with experiments on the Edwards curve Ed25519 with different types of input; namely with 256-bit projective input, with the reduced 255-bit projective coordinates, and finally with affine coordinates. Moreover, we verify the applicability of OTA on Weierstrass curves (Brainpool and NIST) using EM emanations [DPN+16]. A follow-up work from Özgen et al. [OPB16] verified that OTA can successfully give the correct prediction on the scalar bit, by using distinguishers from machine learning (classification methods).

4.2.2 Generic Attack Description

Template attacks consist of two phases, *template building* for characterizing the device and *template matching*, where the characterization of the device together with a

[1] By similar device we mean the same type of microcontroller running the same algorithm.

power trace from the device under attack are used to determine the secret [MOP07]. Therefore, the first condition of our proposed attack is typically fulfilled by all attacks of this kind.

It is well known that template attacks against scalar multiplication can generate templates "on-the-fly", i.e., interleaving the template building and matching phases. See, for example, [MO09, Section 5.3]. We take this idea further by building templates after the target trace has been obtained (condition 2). The attacker, being able to do things in this order, needs only limited control over the target device. Moreover, the attacker is not affected by randomization of the secret data during different executions of the algorithm, since he always has to compare his template traces with the same target trace.

The basic idea consists of comparing the traces for inputs $\boldsymbol{P}$ (target trace) and $2\boldsymbol{P}$ (online template trace) while executing scalar multiplication and then finding similar patterns between them, based on hypothesis on a bit for a given operation. The target trace is obtained only once. For every bit of the scalar, we need to obtain an online template trace with input $[k]\boldsymbol{P}, k \in \mathbb{Z}$, where k is chosen as a function of our hypothesis on this bit. We hereby note that the template trace is part of the target trace (for instance it corresponds to the first doubling) and it is compared bit-by-bit with the target trace. Therefore, alignment of traces is not necessary.

The attack methodology can be summarized as follows:

- Acquire a full target trace from the device under attack, during the execution of a scalar multiplication.
- Locate the doubling and addition operations performed in each round.
- Find multiples of $[m]\boldsymbol{P}$, where $m \in \mathbb{Z}, m \leq k$ and k is the scalar. These points are used to create the template traces.

The methodology offers a generic attack framework, which does not require any previous knowledge of the leakage model nor a specific type of curve. It is applicable to various forms of curves (Weierstrass, Edwards and Montgomery curves), scalar multiplication algorithms and implementations. Contrary to the doubling attack [FV03], OTA can be launched against right-to-left algorithms and the Montgomery ladder.

The main assumption in the OTA attacker model is in his ability to choose an input point to the scalar multiplication algorithm, in order to generate template traces. As it is demonstrated in the original paper, OTA works with one *target trace* from the device under attack and one *template trace* per key bit obtained from the attacker's device that runs the same implementation. Performing OTA in practice requires the following assumptions to be made regarding the attacker:

- The attacker knows the input $\boldsymbol{P}$ of the target device.

- He knows the implementation of the scalar multiplication algorithm and he is able to compute the intermediate values.
- He can choose the input points on a device similar to the target device.

Furthermore, we work with the following assumptions related to the device:

- The scalar can be randomized or blinded.
- The intermediate values are deterministic.

The OTA is then performed as follows:

1. The attacker first obtains a target trace with input point $\boldsymbol{P}$ from the target device.
2. He obtains template traces with input points $[m]\boldsymbol{P},\ m \in \mathbb{Z}$ for multiples of the point $\boldsymbol{P}$, e.g. $2\boldsymbol{P}$ or $3\boldsymbol{P}$.
3. He compares the correlations between the target and each pair of template traces. The correct guess is most likely to be the highest correlation.

The OTA technique, as depicted in Figure 4.1, is originally described for binary algorithms, but it can be easily adapted to the windows method by creating one template for a hypothesis made for each window. The attacker model for OTA is more suitable for the Diffie-Hellman key-exchange protocol, because the input point can be selected. Nevertheless, this attack can be applied against the ECDSA algorithm, if the input point of the target device is known or if the attacker can modify the implementation on the template device and consequently the internal values of the computation. In this case, the attacker could try to recover the ephemeral key k by using x_1, the x-coordinate of $[k]G$ presented in Section 2.3.3, as input to the target. This could be useful to recover the long term secret key of the device under attack, similar to previous template attacks against ECDSA.

At this point, it is important to explain precisely how the interesting points to generate the template traces are chosen. With the term *interesting points* we mean the multiples of the point $\boldsymbol{P}$ that are expected to be the outputs of every iteration of the scalar multiplication algorithm, i.e. $2\boldsymbol{P}$ and $3\boldsymbol{P}$ for the first bit of the scalar. This is demonstrated with a graphical example depicted in Figure 4.1.

Let us assume that the initial input point to the double-and-add-always algorithm is $\boldsymbol{P}$ and the most significant bit (K_{MSB}) of our secret scalar is 1. Then, the output of the second iteration (operations for K_{MSB-1}) is either $2\boldsymbol{P}$ or $3\boldsymbol{P}$. For example, if $K_{MSB-1} = 0$, then the output of the second iteration is $2\boldsymbol{P}$ and consequently the template trace for $2\boldsymbol{P}$ gives higher correlation to the target trace than the template for $3\boldsymbol{P}$. We compute the correlations between the template traces $2\boldsymbol{P}, 3\boldsymbol{P}$, and the target trace, in order to find the most likely key bit. The highest correlation value is considered to be the right key guess.

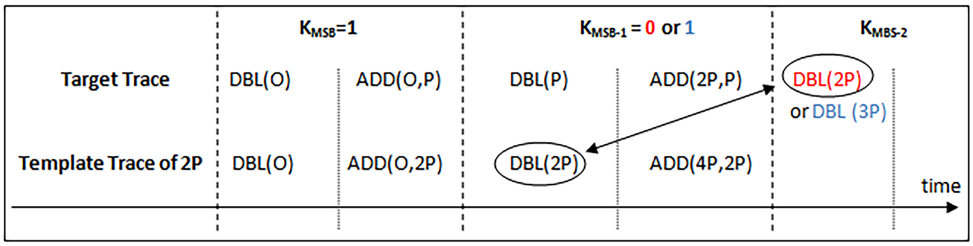

Figure (4.1) How to find the second MSB K_{MSB-1} in the target trace with the template trace of $2\boldsymbol{P}$

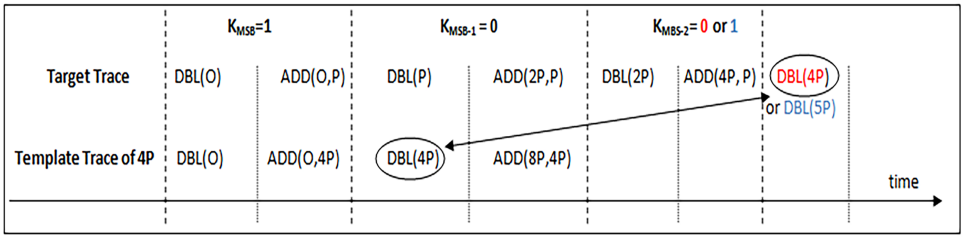

Figure (4.2) How to find the third MSB K_{MSB-2} in the target trace with the template trace of $4\boldsymbol{P}$

We continue the same procedure of calculating the two possible outcomes for bit K_{MSB-2}, which are the template traces for $4\boldsymbol{P}$ or $5\boldsymbol{P}$ (if the previous bit was $2\boldsymbol{P}$), and then finding the highest correlation between the templates and the target trace. Figure 4.2 shows how the templates for the third bit K_{MSB-2} can be generated.

In general, for each iteration of the scalar multiplication algorithm, we compare the second iteration of the scalar multiplication execution (corresponding to the first doubling operation whose consumption is detected with EM) in the template trace with the $(i+1)^{\text{th}}$ execution of the target trace.

In the original paper, pattern matching is performed by using the Pearson correlation coefficient, $\rho(X, Y)$, which measures the linear relationship between two variables X and Y. For power traces, the correlation coefficient shows the relationship between two points of the trace, which indicates the Hamming-weight leakage of key-dependent assignments during the execution of a cryptographic algorithm. Extension to other distinguishers from machine learning is presented in Section 4.2.2; it is the result of our work with Özgen and Batina in [OPB16].

The template matching corresponds to a list of correlation coefficients that show the relationship between all samples from the template trace to the same consecutive amount of samples in the target trace. We performed pattern matching for our traces using an automated module based on the Pearson correlation coefficient, $\rho(X, Y)$, which measures the linear relationship between two variables X and Y. For power

Table (4.1) Two executions of the double-and-add-always algorithm

scalar $k = 4 = (1, 0, 0)$	scalar $k = 5 = (1, 1, 0)$
$R_0 = \boldsymbol{P}$	$R_0 = \boldsymbol{P}$
$R_0 = 2\boldsymbol{P}$, $R_1 = 3\boldsymbol{P}$, return $2\boldsymbol{P}$	$R_0 = 2\boldsymbol{P}$, $R_1 = 3\boldsymbol{P}$, return $3\boldsymbol{P}$
$R_0 = 4\boldsymbol{P}$, $R_1 = 5\boldsymbol{P}$, return $4\boldsymbol{P}$	$R_0 = 6\boldsymbol{P}$, $R_1 = 7\boldsymbol{P}$, return $6\boldsymbol{P}$

traces, the correlation coefficient shows the relationship between two points of the trace, which indicates the Hamming-weight leakage of key-dependent assignments during the execution of a cryptographic algorithm. Extensions to other leakage models and distinguishers are straightforward. If our hypothesis on the given key bit is correct, then the pattern match between our traces at the targeted operation will be high (in our experiments it reached 99%).

In this way we can recover the first i bits of the key. Knowledge of the first i bits provides us with complete knowledge of the internal state of the algorithm just before the $(i+1)^{th}$ bit is processed. Since at least one operation in the loop depends on this bit, we can make a hypothesis about the $(i+1)^{th}$ bit, compute an online template trace based on this hypothesis, and correlate this trace with the target trace at the relevant predetermined point of the algorithm. If this correlation is larger than a certain threshold, the hypothesis is assumed to be correct, otherwise it is assumed to be false.

4.3 OTA on Scalar Multiplication Algorithms

4.3.1 Attacking the Left-to-right Scalar Multiplication Algorithm

The core idea and feasibility of the attack is demonstrated through an example based on the double-and-add-always algorithm described in Algorithm 2. Table 4.1 shows two executions of the algorithm for two different scalars $k = 100$ and $k = 110$. We note that the first execution of the loop always starts by doubling the input point $\boldsymbol{P}$, for all values of k. We assume that $k_{x-1} = 1$. Depending on the second-most significant key bit k_{x-2}, the output of the first iteration of the algorithm will be either $2\boldsymbol{P}$ or $3\boldsymbol{P}$. For any point $\boldsymbol{P}$ we can, therefore, get a power trace for the operation $2\boldsymbol{P}$, i.e. we let the algorithm execute the first two double-and-add iterations. In our setup, we can zoom into the level of one doubling, which will be our template trace. Then we perform the same procedure with $2\boldsymbol{P}$ as the input point to obtain the online template trace that we want to compare with the target trace. If we assume that the second-most significant bit of k is 0, then we compare the $2\boldsymbol{P}$ template with the output of the doubling in the first iteration. Otherwise, we compare it with the online template trace for $3\boldsymbol{P}$.

Assuming that the first $(i-1)$ bits of k are known, we can derive the i^{th} bit by

computing the two possible states of $\boldsymbol{R}_0$ after this bit has been treated and recover the key iteratively. Note that only the assignment in the i^{th} iteration depends on the key bit k_i, but none of the computations do, so it is necessary to compare the trace of the doubling operation in the $(i+1)^{th}$ iteration with the original target trace. To decide whether the i^{th} bit of k is zero or one, the trace that the doubling operation in the $(i+1)^{th}$ iteration would give for $k_{i+1} = 0$ is compared with the target trace. For completeness, we can compare the target trace with a trace obtained for $k_{i+1} = 1$ and verify that it has a lower pattern match percentage; in this case, the performed attack needs two template traces per key bit. However, if during the acquisition phase the noise level is low and the signal is of good quality, an efficient attack can be performed with only our target trace and a single trace for the hypothetical value of $\boldsymbol{R}_{k_{i+1}}$.

4.3.2 Attacking the Right-to-left double-and-add-always Algorithm

In this section we examine the binary right-to-left add-always algorithm of Joye [Joy07], below as Algorithm 8. Contrary to Algorithm 2, the computations in the main loop of the above algorithm clearly depend on the key.

Algorithm 8: Binary right-to-left double-and-add-always algorithm

Input: $\boldsymbol{P}$, $k = (k_{x-1}, k_{x-2}, \ldots, k_0)_2$
Output: $\boldsymbol{Q} = [k]\boldsymbol{P}$

1 $\boldsymbol{R}_0 \leftarrow \mathcal{O}$;
2 $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
3 **for** $i \leftarrow 0$ **up to** *x-1* **do**
4 $\quad b \leftarrow 1 - k_i$;
5 $\quad \boldsymbol{R}_b \leftarrow 2\boldsymbol{R}_b$;
6 $\quad \boldsymbol{R}_b \leftarrow \boldsymbol{R}_b + \boldsymbol{R}_{k_i}$;
7 **end**
8 **return** $\boldsymbol{R}_0$

Attacking the right-to-left double-and-add-always algorithm of [Joy07] is a type of key-dependent assignment OTA. We target the doubling operation and note that the input point will be doubled either in the first (if $k_0 = 0$) or in the second iteration of the loop (if $k_0 = 1$). If k is fixed we can easily decide between the two by inputting different points, since if $k_0 = 1$ we will see the common operation $2\,\mathcal{O}$. If k is not fixed, we simply measure the first two iterations and again use the operation $2\,\mathcal{O}$ if the template generator should use the first or second iteration. Once we are able to obtain clear traces, the attack itself follows the general description of Section 4.2.2. If we assume that the first i bits of k are known and we wish to derive the $(i+1)^{th}$ bit, this means that we know the values of $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$ at the start of the

$(i+1)^{th}$ iteration. By making a hypothesis on the value of the $(i+1)^{th}$ key bit, we can decide according to the matching percentage if $\boldsymbol{R}_0$ or $\boldsymbol{R}_1$ was used.

4.3.3 Attacking the Montgomery Power Ladder

As discussed in Section 2.2.4, the main observation that makes OTA attacks applicable to the Montgomery Power Ladder is that at least one of the computations, namely the doubling in the main loop, directly depends on the key bit k_i. For example, if we assume that the first three bits of the key are 100, then the output of the first iteration will be $R_0 = 2\boldsymbol{P}$. If we assume that the first bits are 110, then the output of the first iteration will be $R_0 = 3\boldsymbol{P}$. Therefore, if we compare the pattern of the output of the first iteration of Algorithm 4 with scalar $k = 100$, we will observe a higher correlation with the pattern of $R_0 = 2\boldsymbol{P}$ than with the pattern of $R_0 = 3\boldsymbol{P}$. This is demonstrated in the working example of Table 4.2.

Table (4.2) Two executions of the Montgomery ladder

scalar $k = 4 = (1,0,0)$		scalar $k = 5 = (1,1,0)$	
$R_0 = \boldsymbol{P}, R_1 = 2\boldsymbol{P}$		$R_0 = \boldsymbol{P}, R_1 = 2\boldsymbol{P}$	
$b_0 = 1 - k_0 = 1$	$R_1 = 3\boldsymbol{P}$, $R_0 = 2\boldsymbol{P}$	$b_0 = 1 - k_0 = 0$	$R_0 = 3\boldsymbol{P}$, $R_1 = 4\boldsymbol{P}$
$b_1 = 1 - k_1 = 1$	$R_1 = 5\boldsymbol{P}$, $R_0 = 4\boldsymbol{P}$	$b_1 = 1 - k_1 = 1$	$R_1 = 7\boldsymbol{P}$, $R_0 = 6\boldsymbol{P}$

4.3.4 Side-channel Atomicity

Algorithms that have the property of side-channel atomicity, as described in Section 2.2.5 are designed, in order to achieve an identical side-channel profile. Simple atomic algorithms do not offer any protection against online template attacks, because the regularity of point operations does not prevent mounting this sort of attack. We recall here Algorithm 9. As long as there is a key-dependent assignment, as the one in line 5, OTA can be mounted.

The point $2\boldsymbol{P}$, as an output of the third iteration of Algorithm 9, will produce a power trace with a pattern that is very similar to the trace that would have the point $2\boldsymbol{P}$ as an input. Therefore, the attack will be the similar to the one described for the binary left-to-right double-and-add-always algorithm; the only difference is that instead of the output of the second iteration of the algorithm, we have to focus on the pattern of the third iteration. In general, when an attacker forms a hypothesis about a certain number of bits of k, the hypothesis will include the point in time where $\boldsymbol{R}_0$ will contain the predicted value. This means that an attacker would have to acquire a larger target trace to allow all hypotheses to be tested.

Algorithm 9: Side-Channel Atomic double-and-add algorithm

Input: $\boldsymbol{P}$, $k = (k_{x-1}, k_{x-2}, \ldots, k_0)_2$
Output: $\boldsymbol{Q} = [k]\boldsymbol{P}$

$R_0 \leftarrow \mathcal{O}$; $R_1 \leftarrow \boldsymbol{P}$; $i \leftarrow x - 1$;
$n \leftarrow 0$;
while $i \geq 0$ **do**
  $\boldsymbol{R}_0 \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_n$;
  $n \leftarrow n \oplus k_i$;
  $i \leftarrow i - \neg n$;
end
return $\boldsymbol{R}_0$

4.4 Practical OTA on Ed25519 implemented on ATMega163

The feasibility and efficiency of OTA is initially shown in [BCP+14, BCP+17] with practical attacks on the double-and-add-always scalar multiplication running on the ATMega163 microcontroller [Cor10] in a smart card. The scalar multiplication algorithm is based on the curve arithmetic of the Ed25519 implementation presented in [HS13]. We hereby give the details of this attack.

4.4.1 Experimental Setup & Tools

Our measurement setup consists of a PC, a Power Tracer and an oscilloscope. The Power Tracer is a low-noise transparent card reader for SCA-DPA side-channel power measurements with precise triggering capabilities [Risb]. For acquiring the traces we used a Picoscope 5203[2] with sampling rate of 125 M samples per second for both target trace and online template traces. For the graphical representation of the traces and analysis of the measurements we used Inspector SCA software [Risa].

The acquisition memory buffer of the Picoscope is limited to 32 M samples. Since 5 iterations of the scalar multiplication algorithm take around 235 *ms*, it means that with sampling rate of 125 M samples per second we can record a trace of approximately 29.4 M samples.

The scalar multiplication algorithm is based on the curve arithmetic of the Ed25519 implementation presented in [HS13].[3] The elliptic curve used in Ed25519 is the twisted Edwards curve $E : -x^2 + y^2 = 1 + dx^2y^2$ with $d = -(121665/121666)$

[2] `http://www.picotech.com/discontinued/PicoScope5203.html`
[3] The code is available online at `http://cryptojedi.org/crypto/#avrnacl`.

and base point

$$\boldsymbol{P} = (151122213495354007725011514095885315114540126930418572060461132839498477622024631683569492647816942839400347516314130799386625622561578303360316525185596 0).$$

For more details on Ed25519 and this specific curve, see [BDL+12].

We modified the software to perform a double-and-add-always scalar multiplication (see Algorithm 2). To illustrate that our attack also works if the template device is not the same as the target device, we used two different smart cards: one to obtain the target trace and one to obtain the online template traces. The whole underlying field and curve arithmetic is the same as in [HS13]. This means in particular that points are internally represented in extended coordinates as proposed in [HWCD08]. In this coordinate system, a point $\boldsymbol{P} = (x, y)$ is represented as $(X : Y : Z : T)$ with $x = X/Z, y = Y/Z$, and $x \cdot y = T/Z$.

4.4.2 Online Template Attack with 256-bit Projective Input

In this section we describe how to apply an OTA if the input supplied to the scalar multiplication is in extended projective coordinates, i.e, if the attacker has full control over all coordinates of the starting point. This is a realistic assumption if a protocol avoids inversions entirely and protects against leakage of projective coordinates by randomization as proposed in [NSS04, Section 6]. Recall that for extended coordinates, T is fully determined by X, Y and Z; they are an extension of standard projective coordinates.

The attack targets the output of the doubling operation. We performed pattern matching for our traces as described in Section 4.2.2. In this way, we could determine the leakage of key-dependent assignments during the execution of the algorithm.

We first demonstrate how to attack a single bit and then we present our results from recovering the five most significant unknown bits of the scalar (recall that the highest bit is always set to one; see Algorithm 2). The remaining bits can be attacked iteratively in the same way as described in Section 4.2.2; as stated above we were not able to recover more than five bits simultaneously due to technical limitations of our measurement setup.

The first observation from our experiments is that when we execute the same algorithm with the same input point on two different cards, there is a constant vertical misalignment between the two obtained traces, but the patterns look almost identical. This fact validates our choice of the correlation coefficient as our pattern-matching metric, since this metric does not depend on the difference in absolute val-

ues and therefore the constant misalignment does not affect the results. Figure 4.3 shows this vertical misalignment between the brown trace obtained from the target and the blue trace obtained from the template device for the same instance of the algorithm.

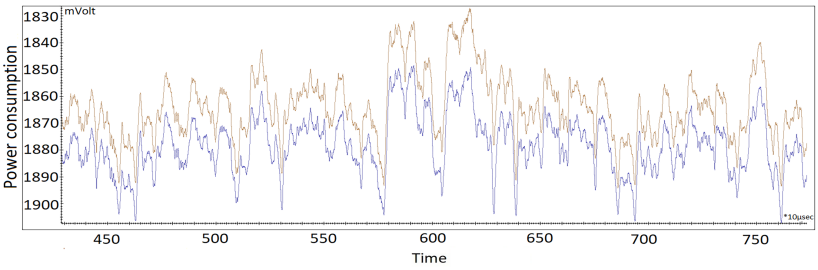

Figure (4.3) Similarity between the traces taken for input $2\boldsymbol{P}$ by the target card and the templates' card

For our target trace, we compute a multiple of a point $\boldsymbol{P}$ with coordinates

$$\begin{aligned}
\boldsymbol{P}_x = \;& \texttt{0x 62 18 E3 09 D4 00 65 FC C3 38 B3 12 7F 46 83 71} \\
& \texttt{82 32 4B D0 1C E6 F3 CF 81 AB 44 E6 29 59 C8 2A,} \\
\boldsymbol{P}_y = \;& \texttt{0x 55 01 49 22 65 E0 73 D8 74 D9 E5 B8 1E 7F 87 84} \\
& \texttt{8A 82 6E 80 CC E2 86 90 72 AC 60 C3 00 43 56 E5,} \\
\boldsymbol{P}_z = \;& \texttt{0x 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00} \\
& \texttt{00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01,} \\
\boldsymbol{P}_t = \;& \texttt{0x 3F C1 7C 25 A0 F7 0F 2B 31 13 A0 5A 48 E6 CD 8B} \\
& \texttt{CD 34 1E 22 9C B1 0E 48 33 B8 19 EA 5D 3A 87 62.}
\end{aligned}$$

We know that the most significant bit of the scalar is 1, so after the first iteration of the double-and-add-always loop the value of $\boldsymbol{R}_0$ is either $2\boldsymbol{P}$ (if the second bit of k is zero) or $3\boldsymbol{P}$ (if the second bit of k is one). We furthermore know from the addition formulas used by the implementation, that $\boldsymbol{R}_0 = 2\boldsymbol{P}$ or $\boldsymbol{R}_0 = 3\boldsymbol{P}$ has the following specific representation in extended coordinates:

$$\begin{aligned}
2\boldsymbol{P}_{\boldsymbol{x}} \;=\; & \texttt{0x B8 30 08 EE B7 49 E5 19 BA 5C 05 E6 3E DA AB A1} \\
& \texttt{E2 BA 0C 92 03 7A 02 79 6B 1D 92 A6 36 A4 97 46,} \\
2\boldsymbol{P}_{\boldsymbol{y}} \;=\; & \texttt{0x 91 0B 93 1F 83 32 56 DB 68 C1 D2 59 71 94 A7 74} \\
& \texttt{97 C4 A9 FA D6 3D 04 25 35 C5 11 84 0C 51 A6 92,}
\end{aligned}$$

$$
\begin{aligned}
2\boldsymbol{P_z} &= \texttt{0x D0 98 E5 67 7B 2A 9C CA 67 82 38 27 9B FC 55 B6} \\
&\quad \texttt{0A 4B 5F 37 74 38 DF 01 5E C2 BF CC 83 B2 B9 22,} \\
2\boldsymbol{P_t} &= \texttt{0x 18 0C 2E 15 36 BA CE 17 B0 96 A0 EE 22 2B 02 99} \\
&\quad \texttt{AA F2 CB EE 86 8C EB 1D 2D 74 80 0E 73 5F 48 D4;}
\end{aligned}
$$

$$
\begin{aligned}
3\boldsymbol{P_x} &= \texttt{0x EA B3 BE 0B 61 DE EB 0B 91 5B 22 8B 3E 00 37 6A} \\
&\quad \texttt{CB 7C 48 71 14 BC B3 4C D9 0A 12 75 BA 58 64 22,} \\
3\boldsymbol{P_y} &= \texttt{0x 23 42 D5 49 33 AF B7 E1 CA 07 9A E7 9E C1 B9 DF} \\
&\quad \texttt{DD 45 D0 CB 96 DE 25 DF 0C 4C 47 4C 52 4B 6E EC,} \\
3\boldsymbol{P_z} &= \texttt{0x A9 C7 59 0B 5B 80 3C 2E AB 6B AD E9 7E A9 C3 31} \\
&\quad \texttt{1A E8 3B C9 8D 65 9A E1 3A 9D 4D 0A D6 F9 3D 2A,} \\
3\boldsymbol{P_t} &= \texttt{0x 4A 9D 4C 7F 68 7A 53 B2 1C FD 06 DB 14 00 B1 EA} \\
&\quad \texttt{7A A4 43 4D E9 04 EF 26 24 D0 01 F4 9B 49 14 34.}
\end{aligned}
$$

To determine the second bit of the secret scalar k, we generate template traces by inputting exactly those representations of $2\boldsymbol{P}$ and $3\boldsymbol{P}$ and computing the correlation of the first iteration of the template trace with the second iteration of the target trace. We use the fact that inputs are given in projective representation.

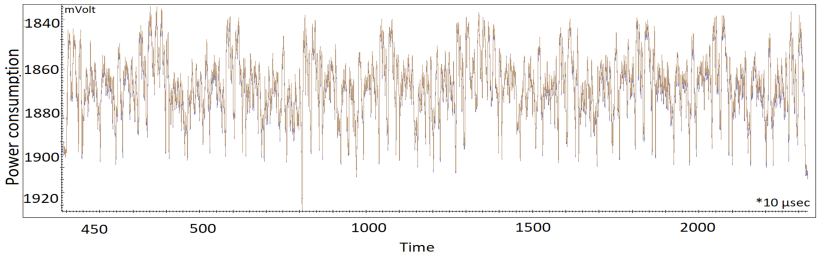

Figure (4.4) Comparison between the trace that comes from input $\boldsymbol{P}$ (target trace, brown) at second iteration and the trace with input $2\boldsymbol{P}$ (matching template trace, blue) at first iteration. For illustration, both traces were obtained from the same card to avoid vertical misalignment.

In fact, we will see that the correlation between the correct template trace and the target trace is so much higher than between the wrong template trace and the target trace, that just one of the two template traces is sufficient to determine the second bit of k. All subsequent figures are taken with the six most significant bits of k set to 100110.

Figure 4.4 shows power traces of the second iteration of the target trace (brown) and the first iteration of the $2\boldsymbol{P}$ template trace, i.e., the matching template trace.

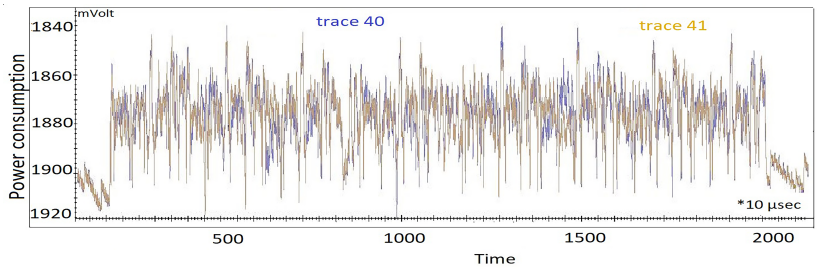

Figure (4.5) Comparison between $\boldsymbol{P}$ (target trace, brown) at second iteration and $3\boldsymbol{P}$ (non-matching template trace, blue) at first iteration. For illustration, both traces were obtained from the same card to avoid vertical misalignment.

Figure 4.5 shows power traces of the second iteration of the target trace (brown) and the first iteration of the $3\boldsymbol{P}$ template trace, i.e., the non-matching template trace.

For validation of our results, we conducted several experiments with different input points from the target card and the template card, and computed the correlation in the obtained power traces. Figure 4.6 shows the correlation of the template trace (iteration 1) with input $2\boldsymbol{P}$ to the target trace (iteration 2) in blue and the correlation of the template trace (iteration 1) with input $3\boldsymbol{P}$ to the same target trace (iteration 2) in brown. We notice that the trace obtained from the point $2\boldsymbol{P}$ is almost identical to the pattern obtained from the target trace; as expected the correlation is at least 97% for all our experiments. On the other hand, the correlation of the target trace with the template trace for $3\boldsymbol{P}$ is at most 83%. To determine the value of one bit, we can thus simply compute only one template trace, and decide the value of the targeted bit depending on whether the correlation is above or below a certain threshold set somewhere between 83% and 97%.

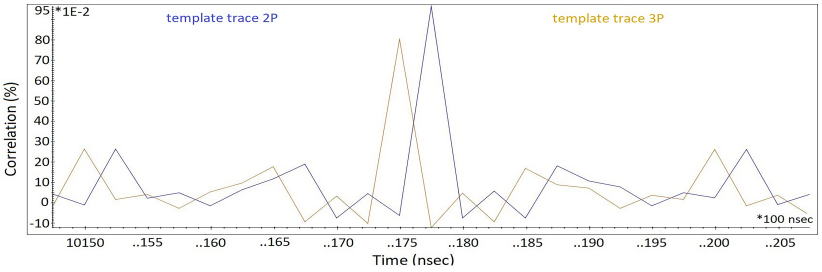

Figure (4.6) Pattern matching percentage of $2\boldsymbol{P}$ to $\boldsymbol{P}$ and $3\boldsymbol{P}$ to $\boldsymbol{P}$ for trace with $\boldsymbol{P}$ for different cards for template and target traces

Whether we use the same card for target and template traces or different cards, the results in the correlation do not change. When we use a different card, we notice a misalignment in the peaks as in Figure 4.6 that disappears if the traces are taken

from the same card as in Figure 4.7. However, the correlation values are the same for both situations.

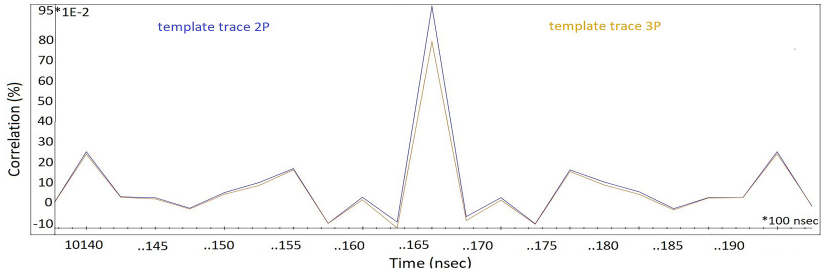

Figure (4.7) Pattern matching percentage of $2\boldsymbol{P}$ to $\boldsymbol{P}$ and $3\boldsymbol{P}$ to $\boldsymbol{P}$ for trace with $\boldsymbol{P}$ obtained from the same card for template and target traces

The results presented so far are obtained while attacking a single bit of the exponent. When we attack five bits with one acquisition, we observe lower numbers for pattern matching for both the correct and the wrong scalar guess. The correlation results for pattern matching are not so high, mainly due to the noise that is occurring in our setup during longer acquisitions. This follows from the fact that our power supply is not perfectly stable during acquisitions that are longer than 200 *ms*. However, the difference between correct and wrong assumptions is still remarkable as depicted in Figures 4.8 to 4.12, showing the OTA on five scalar bits $k = 100110$ at once. Correct bit assumptions have $84 - 88\%$ matching patterns, while the correlation for the wrong assumptions drops to $50 - 72\%$. To determine the value of one bit, it is thus necessary to compute only one template trace, and decide on the value of the targeted bit depending on whether the correlation is above or below a certain threshold (in this case, the threshold can be set to 80%). The template traces are always taken during the first iteration and the target trace contains all five iterations. Note that the input points for the template traces depend on the already recovered bits of the exponent.

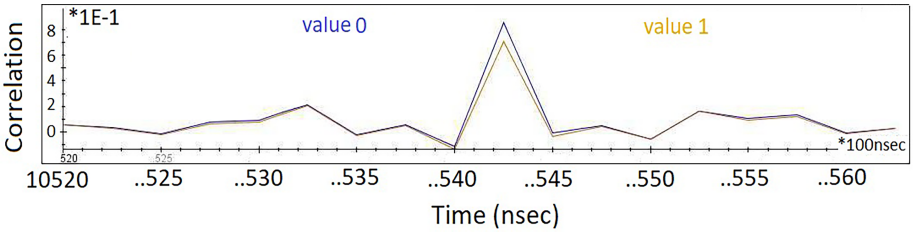

Figure (4.8) Pattern matching percentage of $2\boldsymbol{P}$ to $\boldsymbol{P}$ and $3\boldsymbol{P}$ to $\boldsymbol{P}$ obtained for the 2nd MSB of the scalar

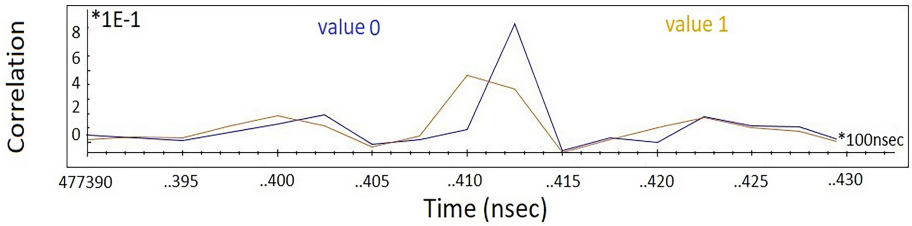

Figure (4.9) Pattern Matching $4\boldsymbol{P}$ to $\boldsymbol{P}$ and $5\boldsymbol{P}$ to $\boldsymbol{P}$ obtained for the 3rd MSB of the scalar

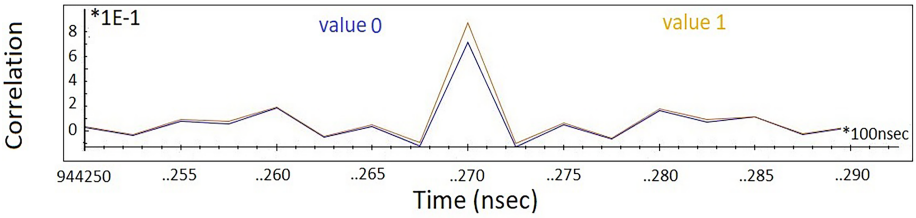

Figure (4.10) Pattern Matching $8\boldsymbol{P}$ to $\boldsymbol{P}$ and $9\boldsymbol{P}$ to $\boldsymbol{P}$ obtained for the 4th MSB of the scalar

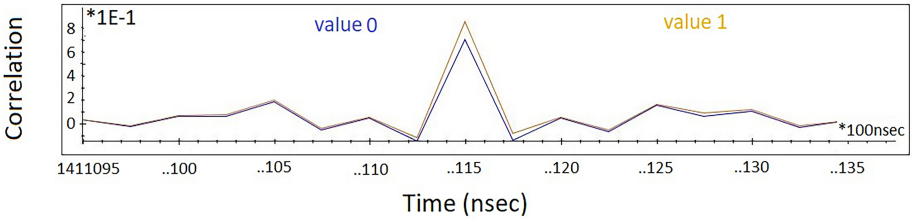

Figure (4.11) Pattern Matching $18\boldsymbol{P}$ to $\boldsymbol{P}$ and $19\boldsymbol{P}$ to $\boldsymbol{P}$ obtained for the 5th MSB of the scalar

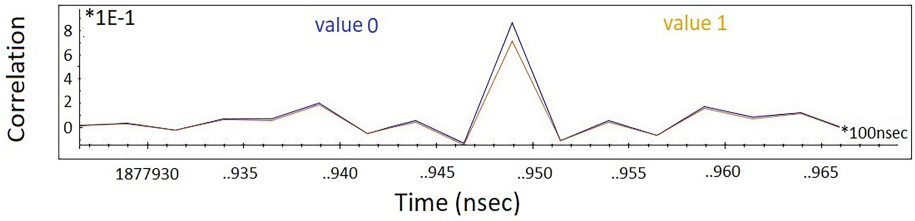

Figure (4.12) Pattern Matching $38\boldsymbol{P}$ to $\boldsymbol{P}$ and $39\boldsymbol{P}$ to $\boldsymbol{P}$ obtained for the 6th MSB of the scalar

Note that the attack with projective inputs does not make any assumptions on formulas used for elliptic curve addition and doubling. In fact, we carried out the attack for specialized doubling and for doubling that use the same unified addition formulas as addition. The results were similar: all traces shown above are from the experiments that used unified addition formulas for both addition and doubling.

4.4.3 Online Template Attack with 255-bit Projective Input

In the previous section, for simplicity, we deliberately ignored the case of coordinate reduction in the field, in order to make the concept of the attack clear. The implementation that we attack, for the sake of efficiency, operates on 256-bit coordinates and not 255-bit coordinates from the field $\mathbb{F}_{2^{255}-19}$. The 256-bit coordinates correspond to the coordinates from $\mathbb{F}_{2^{255}-19}$ by applying the modulo $2^{255}-19$ operation; by using 256 bits the implementation can save time by not performing some modulo operations.

So far we assumed that we can send to the card the optimized 256-bit coordinates. It is interesting to examine a more complex attack scenario in which we can only input the 255-bit coordinates. In this section, we show that OTA is successful in this scenario as well; a fact that makes OTA a powerful attack technique independent of the prime p of the field used. Fast modular reduction is implemented in [HS13] by using simple shifts and additions, which are relatively cheap on AVRs.

Our idea is not to attack a whole doubling operation, but just a single squaring. More precisely, we show how to perform OTA on the squaring operation of the Z coordinate. First, let us consider the old 256-bit templates. There are two distinct attack cases, namely:

- *MSB* $= 0$ for the Z projective coordinate (the remaining coordinates can have the most significant bit equal to 1), therefore the Z coordinate after reduction remains the same.
  In this case, OTA can be applied in a similar way as in Section 4.4.2. We take the old template coordinates and perform a reduction of all the coordinates modulo our prime number $2^{255}-19$; then we send those coordinates as input to the card to obtain the new templates.

- The Z coordinate has *MSB* $= 1$ and therefore, there is a 9 bit difference from the corresponding $256 - bit$ coordinate.
  In this case, the reduced point differs from its $256 - bit$ equivalent in the MSB and in the least significant byte due to the pseudo-Mersenne prime that we use (i.e., $2^{255}-19$). This case is the most interesting and we will analyze it in the remaining part of this section.

We focus on Step D: the computation of Z^2 (see Figure 4.14 for the details about the doubling formula that we use); we choose a new Z', such that $MSB_{Z'} = 0$ and the rest of the bits are the same as the Z coordinate of the old template.

So our new point has only 1-bit difference with the original target trace. For this Z', we recalculate X', Y' and T' using the following equations:

$$\begin{aligned} X' &= X \cdot Z', \\ Y' &= Y \cdot Z', \\ T' &= T/Z', \end{aligned}$$

where X, Y, Z, and T denote the coordinates of the old template.

The results obtained from this attack are similar to the previous section. Figure 4.13 presents the pattern match between a template trace during computation of $D \leftarrow Z^2$ with template with 1-bit difference, 9-bit difference, or wrong template (iteration 1) to the target trace (iteration 2). As expected, the highest peak corresponds to the template with only 1-bit difference, the slightly smaller peak correspond to the 9-bit difference, and the lowest peak corresponds to a wrong template.

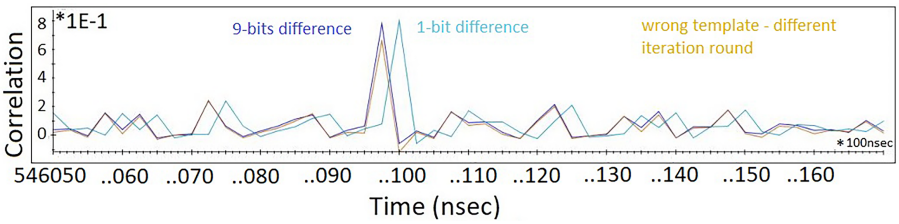

Figure (4.13) Pattern Matching during computation of D of a template with 1-bit difference, a template with 9-bit difference, and a wrong template to the target trace for area of computing D.

Successful key guesses (for the templates that have at most 1-bit difference) give correlation values between 81% and 86%, while unsuccessful ones are below 76%. The success and failure rates are different from those in Section 4.4.2 because now we concentrate on a single squaring and not the whole doubling.

4.4.4 Online Template Attack with Affine Input

The attack as explained in the previous sections makes the assumption that the attacker has full control over the input in projective coordinates. Most implementations of ECC use inputs in affine (or compressed affine) coordinates and internally convert to projective representation. The input is now given as (x, y) and at the beginning of the computation converted to $(x : y : 1 : xy)$. We observe that the points $\boldsymbol{P}, 2\boldsymbol{P}$ and $3\boldsymbol{P}$ do not have any coordinates in common with the projective representations used internally. Already after the first iteration of the double-and-add-always loop, $Z = 1$ does not hold anymore. Those attacks are more elaborate, since the internal point representation changes at every step of the algorithm.

First we consider an attack on the second most significant bit (which is again set to zero) and the input point $\boldsymbol{P}$ of the target trace with coordinates

$$\begin{aligned}\boldsymbol{P}_x =& \texttt{0x 62 18 E3 09 D4 00 65 FC C3 38 B3 12 7F 46 83 71}\\ & \texttt{82 32 4B D0 1C E6 F3 CF 81 AB 44 E6 29 59 C8 2A,}\\ \boldsymbol{P}_y =& \texttt{0x 55 01 49 22 65 E0 73 D8 74 D9 E5 B8 1E 7F 87 84}\\ & \texttt{8A 82 6E 80 CC E2 86 90 72 AC 60 C3 00 43 56 E5.}\end{aligned}$$

Choosing the affine versions of $2\boldsymbol{P}$ and $3\boldsymbol{P}$ to generate template traces does not help us now because they do not have any coordinates in common with the projective representations used internally.[4] To successfully perform the attack we need to modify our approach and take a closer look at the formulas used for point doubling. We illustrate the approach with the unified doubling formula from [HWCD08].[5].

$$\begin{aligned}1: &\quad A \leftarrow (Y_1 - X_1)(Y_2 - X_2)\\ 2: &\quad B \leftarrow (Y_1 + X_1)(Y_2 + X_2)\\ 3: &\quad C \leftarrow kT_1T_2\\ 4: &\quad D \leftarrow 2Z_1Z_2\\ 5: &\quad E \leftarrow B - A\\ 6: &\quad F \leftarrow D - C\\ 7: &\quad G \leftarrow D + C\\ 8: &\quad H \leftarrow B + A\\ 9: &\quad X3 \leftarrow EF\\ 10: &\quad Y3 \leftarrow GH\\ 11: &\quad T3 \leftarrow EH\\ 12: &\quad Z3 \leftarrow FG\end{aligned}$$

Figure (4.14) Unified addition/doubling formula from [HWCD08]

[4]This property follows from the fact that the Z coordinate of $2\boldsymbol{P}$ during the conversion to extended coordinates is always set to `0x01` while the Z coordinate of the point P after being squared in the first iteration of the scalar multiplication loop does not equal `0x01` with overwhelming probability

[5]For details, see: `http://www.hyperelliptic.org/EFD/g1p/auto-twisted-extended-1.html#addition-madd-2008-hwcd-3` These formulas contain the operations listed in Figure 4.14

The main idea of the attack is to focus on the first multiplication $(Y_1 - X_1)(Y_2 - X_2)$, which in case of a doubling would be $(Y - X)^2$. We give now a detailed description on how to generate the necessary templates for $(Y - X)^2$.

Let us assume that the target trace with the point $\boldsymbol{P}$ is already acquired and that we attack bit b_i, where $0 \leq i < x$. Firstly, depending on already recovered bits of the scalar $b_x, \ldots b_{i+1}$ (at the beginning we only know that the most significant bit b_x is 1), the coordinates of P, and the bit guess $b_i \in \{0, 1\}$, we can compute the intermediate value $\lambda = Y - X$ that is squared in Step 1 (Figure 4.14) during acquisition of the target trace. Secondly, we search for a new point $\boldsymbol{P^i} = (x', y')$ such that $Y' - X' = \lambda$.[6] Such a point $\boldsymbol{P^i}$ cannot always be found on the curve, but we can flip the least significant bit of λ and check whether this point belongs to the curve. If this fails, we flip the bit back and then flip the second least significant bit of λ; we continue this way with subsequent least significant bits until we find a point on the curve. From our experiments, we succeed in finding a point on the curve in a maximum of five trials. Such a point will differ from λ on at most 1 bit (in least significant byte of the coordinate).

Using the method described above we compute two points $\boldsymbol{P[k_{x-2} = 0]}$ and $\boldsymbol{P[k_{x-2} = 1]}$, where k_{x-2} indicates the second most significant bit.

$$\begin{aligned}
\boldsymbol{P[k_{x-2} = 0]_x} &= \texttt{0x2B1FDBA73C0BB44A21D59EE599B66E5B} \\
&\quad \texttt{470EA5ADB62777A55254E646F0ADE032}, \\
\boldsymbol{P[k_{x-2} = 0]_y} &= \texttt{0x03FB65D807F4260BD03B6B58CC706A2D} \\
&\quad \texttt{FC19431688EA79511CFC6524C65AEF7C}, \\
\boldsymbol{P[k_{x-2} = 1]_x} &= \texttt{0x340C1C83C144EA9C5B707C5081FD770C} \\
&\quad \texttt{F6C2C95FC044604B45ABAEF3F4F3EDAE}, \\
\boldsymbol{P[k_{x-2} = 1]_y} &= \texttt{0x6D9B33C19315B772941CF4ACE2BEF982} \\
&\quad \texttt{088C51BA4265D2DD78EDE3CA8CE6F852}.
\end{aligned}$$

Similarly to Section 4.4.2, let us assume that the six most significant bits of the scalar k are set to 100110 (recall that the most significant bit is always set to 1). When we compare the trace for $\boldsymbol{P}$ as input at the second iteration to the trace for $\boldsymbol{P[k_{x-2} = 0]}$ at the first iteration during the second squaring operation (computing A) then we can observe that the two traces are almost identical, see Figure 4.15 for details. A is depicted in the highlighted part. The part of the brown trace that is different from the target trace at 10 msec is due to the fact that $Z_{\boldsymbol{P[k_{x-2}=0]}}$ differs from the coordinate $Z_{\boldsymbol{P}}$ during the computation of $D = Z_1 Z_2$.

Figure 4.16 shows the pattern match between a template trace during computation of $A \leftarrow (Y - X)^2$ with input point $\boldsymbol{P[k_{x-1} = 0]}$ (iteration 1) to the target trace

[6] Note that we cannot use $\boldsymbol{P}(X, Y, Z)$ "freely", because now $Z \neq 1$.

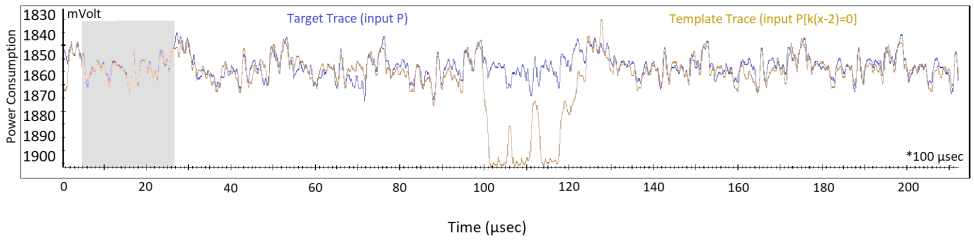

Figure (4.15) Comparison between $\boldsymbol{P}$ at the second iteration to $\boldsymbol{P[k_{x-2}=0]}$ at first iteration; the area of computing A is highlighted

for $\boldsymbol{P}$ (iteration 2) and the pattern match between the template trace (iteration 1) for $\boldsymbol{P[k_{x-1}=1]}$ to the target trace (iteration 2). We notice that the trace obtained from the point $\boldsymbol{P[k_{x-2}=0]}$ is almost identical to the pattern obtained from the target trace as expected. The correlation is 86% for the correct key-guess and under 73% for the incorrect one.

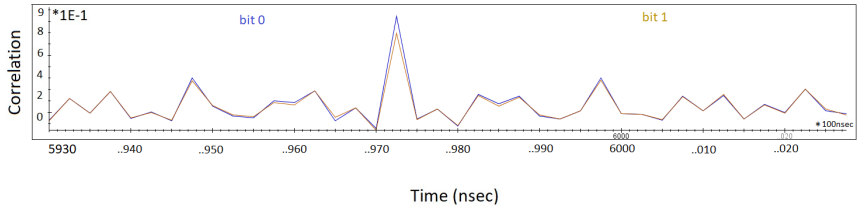

Figure (4.16) Pattern Matching during computation of A of $\boldsymbol{P[k_{x-2}=0]}$ to $\boldsymbol{P}$ and $\boldsymbol{P[k_{x-2}=1]}$ to $\boldsymbol{P}$ with $\boldsymbol{P}$ obtained for the 2nd most significant scalar bit

Since we know the two most significant bits, we can repeat the attack for the next 4 most significant bits (in total we will know the 6 most significant bits since the most significant is always 1). Using the same method for computing the points $\boldsymbol{P[k_{x-2}=0]}$ and $\boldsymbol{P[k_{x-2}=1]}$, we compute the point for the 8 subsequent template points. The pattern matching results for the templates are presented in Figure 4.17, Figure 4.18, Figure 4.19, and Figure 4.20.

The correlation is $84-87\%$ for the correct key guesses. For the non-matching template point the correlation value of the matching patterns is at most 73%. Once again, the threshold value of 80% is verified.

4.5 Practical OTA on software implementation of mbedTLS

The applicability of Online Template Attacks was extended to attacks on scalar multiplication of Brainpool [LMSS14] and NIST [Nat09] curves. In [DPN+16], we

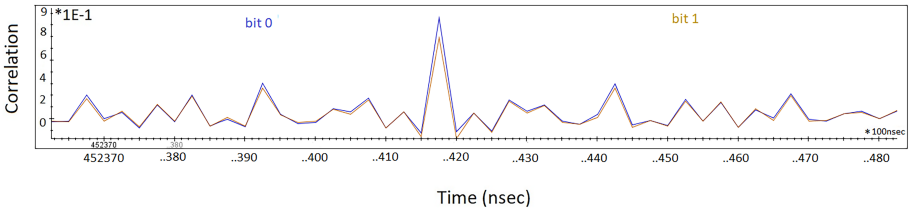

Figure (4.17) Pattern Matching during computation of A of $\boldsymbol{P[k_{x-3}=0]}$ to $\boldsymbol{P}$ and $\boldsymbol{P[k_{x-3}=1]}$ to $\boldsymbol{P}$ with $\boldsymbol{P}$ obtained for the 3rd most significant scalar bit the area of computing A

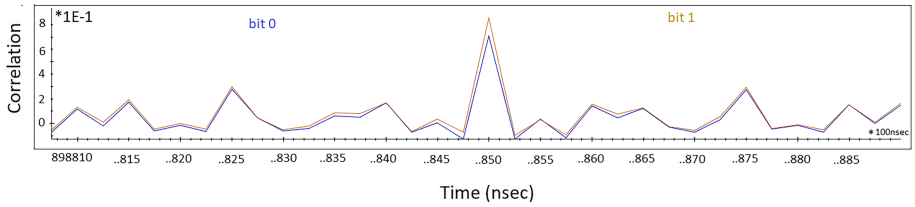

Figure (4.18) Pattern Matching during computation of A of $\boldsymbol{P[k_{x-4}=0]}$ to $\boldsymbol{P}$ and $\boldsymbol{P[k_{x-4}=1]}$ to $\boldsymbol{P}$ with $\boldsymbol{P}$ obtained for the 4th most significant scalar bit the area of computing A

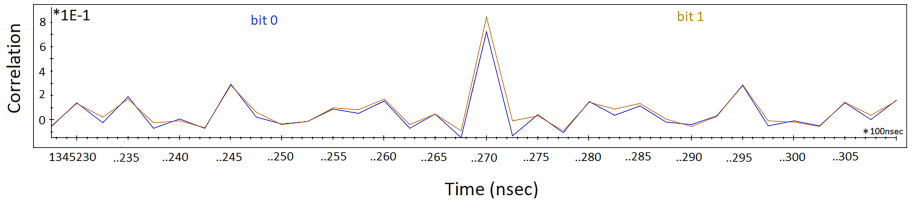

Figure (4.19) Pattern Matching during computation of A of $\boldsymbol{P[k_{x-5}=0]}$ to $\boldsymbol{P}$ and $\boldsymbol{P[k_{x-5}=1]}$ to $\boldsymbol{P}$ with $\boldsymbol{P}$ obtained for the 5th most significant scalar bit the area of computing A

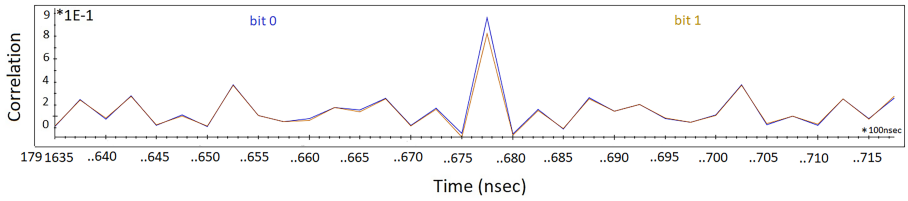

Figure (4.20) Pattern Matching during computation of A of $\boldsymbol{P}[\boldsymbol{k_{x-6}} = \mathbf{0}]$ to $\boldsymbol{P}$ and $\boldsymbol{P}[\boldsymbol{k_{x-6}} = \mathbf{1}]$ to $\boldsymbol{P}$ with $\boldsymbol{P}$ obtained for the 6th most significant scalar bit the area of computing A

propose a generic method to distinguish matching templates and by using two templates per key bit, we manage to detect and correct errors for wrong bit assumptions. This fact increases the success rate of the adaptive template attack significantly compared to the original OTA, reaching 99.8% when 100 average template traces are used. We also take advantage of the horizontal and vertical leakage, which occurs in the broadly used software implementation of mbedTLS during the modular multiplication of large integers (256-bit elements).

As mentioned earlier, the scalar multiplication is the main operation in cryptographic protocols using ECC, such as ECDSA signatures [AX98] or the Diffie-Hellman key-exchange protocol (ECDH) [Nat09]. Many scalar multiplication algorithms are used for efficiency and/or resistance against side-channel attacks. In this section, we show an attack against the binary left-to-right double-and-add-always algorithm (see [Cor99, Joy03]), which is considered to be resistant against simple power analysis (SPA). Our attack applies to other regular algorithms as well, similarly to the original OTA [BCP+14].

4.5.0.1 Scalar multiplication module of mbedTLS

The cryptographic library mbedTLS, formerly known as PolarSSL, is open-source [mbe] and it was acquired by ARM. MbedTLS contains C and assembly code to speed up elliptic curve computations over the chosen finite field. The source code is nicely decomposed into modular blocks and it can be used in embedded devices. For ECC operations, the module `ecp` of the library is used.

MbedTLS is intended to be used in embedded systems which include a hardware multiplier, like smart-phones. Scalar multiplication in mbedTLS consists of two steps: multiplication and modular reduction. Faster scalar multiplication is achieved by using the doubling operation in Jacobian coordinates (DBL) and mixed addition [CMO98] between a point in Jacobian and a point in affine coordinates (ADD). The cost of these operations is explained and detailed by Bernstein and Lange in [BL].

The double-and-add-always algorithm takes as input a point $\boldsymbol{P} = (x_{\boldsymbol{P}}, y_{\boldsymbol{P}})$ in

affine coordinates and the scalar k. For our experiments the scalar is 256-bit long. For every iteration the computation block performs a doubling operation and an addition with $\boldsymbol{P}$.

In PolarSSL v1.3.7 and mbedTLS v2.2.0 [7] a multiplication between two elements in the finite field is computed as described in Algorithm 10. The result of the multiplication is stored in a 512-bit element, called *"multiplication-before-reduction"*; then the result is reduced modulo p (the characteristic of the finite field). For our experiments, we used the Brainpool curve brainpoolP256r1 recommended by BSI [BSI10] and the NIST curve P-256 recommended by NIST [NIS13]. These curves are defined over a 256-bit field and have security level of 128 bits (see [BCC+12] for more details). One element in the finite field of those curves has a length of 256 bit.

The micro-controller used for the experiments is a Cortex-M4 with 32-bit registers for data operations (see Section 4.5.3.1 for more details). Therefore, one field element corresponds to 8 words of 32 bits.

Algorithm 10: Multiplication in mbedTLS

Input: A and $B_7..B_0$ two elements of 256-bits long.
Output: $X = A \times B$

1: $X \leftarrow 0$
2: **for** i from 7 down to 0 **do**
3: $\quad (C, X_{i+7}, X_{i+6}, \ldots, X_i) \leftarrow (X_{i+7}, \ldots, X_i) + A \times B_i$
4: $\quad j \leftarrow i + 8$
5: $\quad$ **repeat**
6: $\quad\quad (C, X_j) \leftarrow X_j + C$
7: $\quad\quad j \leftarrow j + 1$
8: $\quad$ **until** $C \neq 0$
9: **end for**
10: return X

Let A and B be two 256-bit elements in the finite field. Then, A (resp. B) can be written as 8 words A_i for all $i \in \{0, 1, \ldots, 7\}$ (resp. B_i) of 32-bits. A_0 is the least significant word (LSW) of A and A_7 is the most significant word (MSW) of A. Let X be the result of the multiplication $A \times B$ before reduction; X can be represented by 16 words of 32 bits ($X_{15}X_{14} \ldots X_0$).

[7] When we started this project in November 2014, we initially used PolarSSL v1.3.7. During this month PolarSSL was acquired by ARM Holdings. Therefore, we tested later our results in mbedTLS v2.2.0 and they were the same.

4.5.1 Horizontal Leakage due to Propagation of Carry

Horizontal leakage usually occurs when there are conditional statements in the algorithm. This is the case for PolarSSL v1.3.7, mbedTLS v2.2.0 and OpenSSL v1.0.2 and earlier versions. For mounting our attack, we focus in the doubling operation inside the scalar multiplication. This is the interesting operation that we trigger and create our templates from.

In case the whole doubling operation is used to construct templates, it is not possible to achieve high similarity between our templates and the target, mainly due to the noise and the non-constant time implementation. As explained later in Section4.5.3.3, we cannot use the intermediate values (in Jacobian coordinates) as input point (in affine) for the templates. However, by focusing on the operations in the first doubling of the double-and-add-always algorithm to construct the template traces, we achieve more accurate results. For the template pattern, we need only the pattern of the first finite-field multiplication in the doubling.

In the implementation we used (Algorithm 13 in [Riv11]), the first operations during the doubling of point $\boldsymbol{P} = (X, Y, Z)$ are the following:[8]

$$\begin{array}{rcll} D_1 & \leftarrow & X \times & X \mod p \\ D_2 & \leftarrow & Y \times & Y \mod p \\ & & \vdots & \end{array} \tag{4.1}$$

Algorithm 10 shows how multiplication is performed in mbedTLS. The result $A \times B_i$ is stored in eight 32-bit words and there is a potential carry C, which needs to be stored separately (see step 3). This potential overflow creates a significant pattern that can be distinguished from its high amplitude when $C = 1$; this pattern is the carry propagation as depicted in Figure 4.21.

The leakage due to the propagation of the carry depends on the MSW of the input data A_7. For BrainpoolP256r1, $\max\{A_7 | A \in \mathbb{F}_p\} = \texttt{0xA9FB57DA}$ and the probability of having a carry propagation is close to $p = 0.17$. For P-256, $\max\{A_7 | A \in \mathbb{F}_p\}$ equals $2^{32} - 1$, so this probability is close to $p = 0.25$. As shown in Figure 4.21, we can have 7 carry propagations during the multiplication, but we cannot detect the last propagation. So, the probability to have two templates with the same carry propagation, denoted by $\mathbb{P}(C)$, is:

$$\mathbb{P}(C) = \sum_{i=0}^{6} \binom{6}{i} p^{2i} (1-p)^{2(6-i)}, \tag{4.2}$$

where p is the probability to have an internal carry propagation. The probability to have horizontal leakage is 0.86 using $p = 0.17$ for brainpoolP256r1. For P-256, the probability to have horizontal leakage is 0.95 using the fact that $p = 0.25$.

[8] The beginning of the doubling operation is the implementation in PolarSSL v1.3.7. The sequence of the finite field operations in the doubling operation in the mbedTLS v2.2.0 changes to: $D_1 \leftarrow X \times X, D_2 \leftarrow 3 \times X$, but this does not affect the efficiency of our attack.

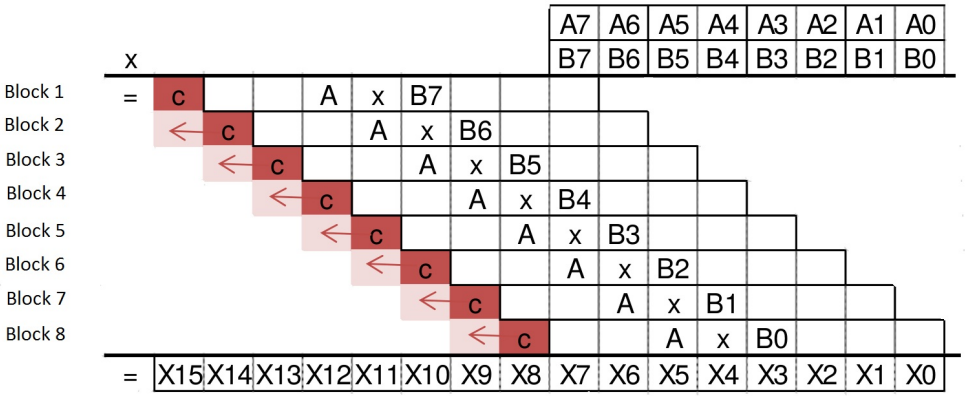

Figure (4.21) Propagation of carry during multiplication in the field

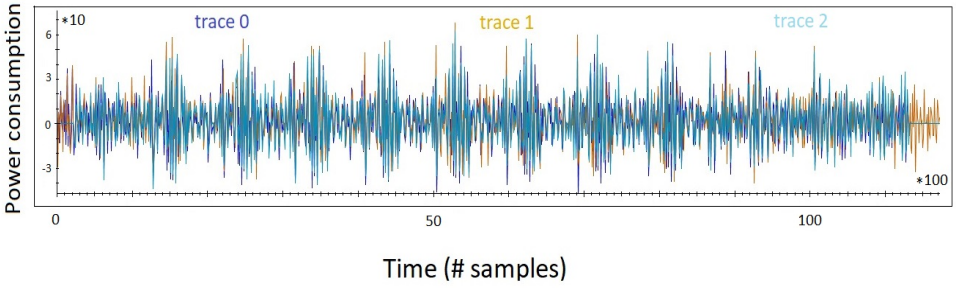

Figure (4.22) No carry propagation between target and template traces

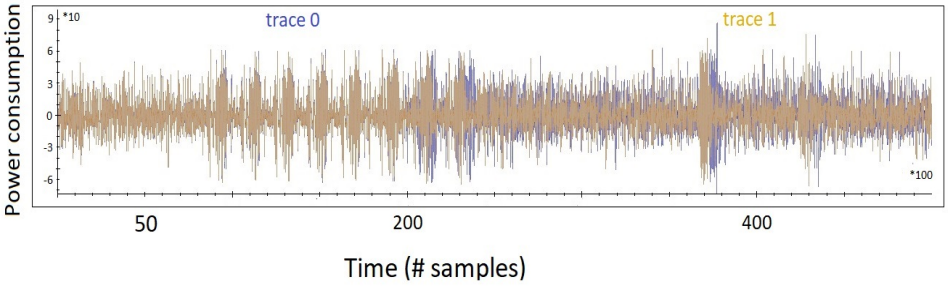

Figure (4.23) Propagation of carry between target and template traces

However, it is more interesting from the OTA point of view to find out when a difference in the carry propagation occurs between the target and template traces. This is the only part of mbedTLS that is non-constant time and we take advantage of this timing difference, every time it occurs. In this case, there is an obvious horizontal leakage between the target and the template traces, as depicted in Figures 4.22, 4.23.

In Figure 4.24, we see how the squaring operation in the beginning of the dou-

bling looks like, when carry propagation occurs.

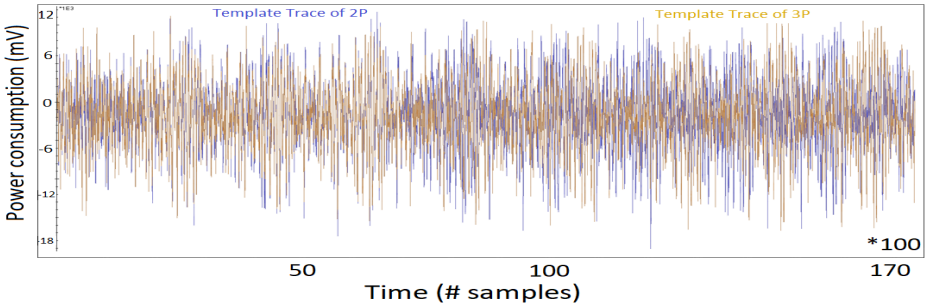

Figure (4.24) Squaring of two random data with different carry propagation

4.5.2 Vertical Leakage due to Signal Amplitude

In constant time executions of our implementation, there is no difference in the carry propagation and the template traces are synchronized with the target trace. In those cases, we observe only a vertical leakage due to the amplitude of the signal and the same method as described in [BCP+14] can be used. To observe this leakage, we use the pattern matching technique using the Pearson correlation as a distinguisher.

4.5.3 Detailed Phases of the Attack in Practice

4.5.3.1 Experimental Setup

The target device is an STM32F4 micro-controller, which contains an ARM Cortex-M4 processor running at its maximum frequency (168 MHz). We have ported the assembly code originally included in PolarSSL v1.3.7 to ARM Cortex-M4 and implemented the double-and-add-always procedure as described in [Cor99, Joy03]. For the acquisition, we used a 54855 Infiniium Agilent oscilloscope and a Langer EMV-TECHNIK RF-U5-2 near field probe. The sampling frequency is 1 GSa/s with 50 MHz hardware input low-pass filter enabled. Matlab 2014b is used for the analysis, and Inspector SCA tool [Risa] for depicting the traces in this chapter. The position of the probe was determined to maximize the signal related to the activity of the 32×32 hardware multiplier.[9]

For the curves defined in Section 4.5, one element in the finite field has a length of 256 bits. Thus, each operation over the field consists of manipulating eight processor words (8×32 bits). In our implementation, a multiplication-before-reduction consists of eight multiplications between a 256-bit element by each 32-bit word of the second element. It leads to eight easily identifiable patterns of eight blocks in

[9] This is a simple identification phase, where we scan the device and find where the crypto processor is. Then we just move the probe around this position, in order to get a signal that is clear as possible.

the EM traces. The length between two blocks can be different depending on the carry propagation, as explained in Section 4.5.1.

4.5.3.2 Preprocessing Phase

The preprocessing phase starts with choosing an input point $\boldsymbol{P}$ and obtaining the target trace from our target device; this is depicted in Figure 4.25. In this trace,

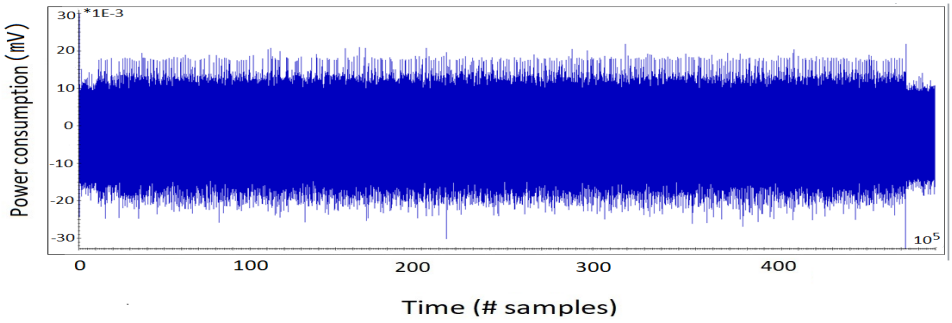

Figure (4.25) EM acquisition for scalar multiplication on P-256 with $k = \texttt{0xA5A5}$

we need to spot the multiplication patterns, which are eight blocks of 256×32 multiplications depicted in Figure 4.26. We note here that this does not constitute a building phase in the usual template setting, it is just an identification phase. From the implementation and the device, we know that a 256-bit element is processed by 32-bit multipliers. Therefore, we expect to see eight patterns for each multiplication. The multiplication procedure is described in Section 4.5.0.1.

When we have a clear pattern for the multiplication, we cross-correlate this pattern with our target trace and we obtain the cross-correlation pattern with one peak at the position of every multiplication. Figure 4.27 shows the cross-correlation of the target trace with the multiplication pattern. By counting the peaks in the cross-correlation trace, we can find the part of the computation that we are interested in.

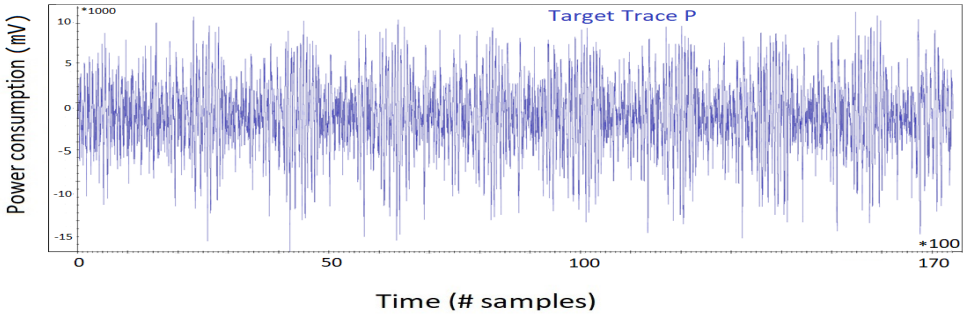

Figure (4.26) Pattern of multiplication-before-reduction

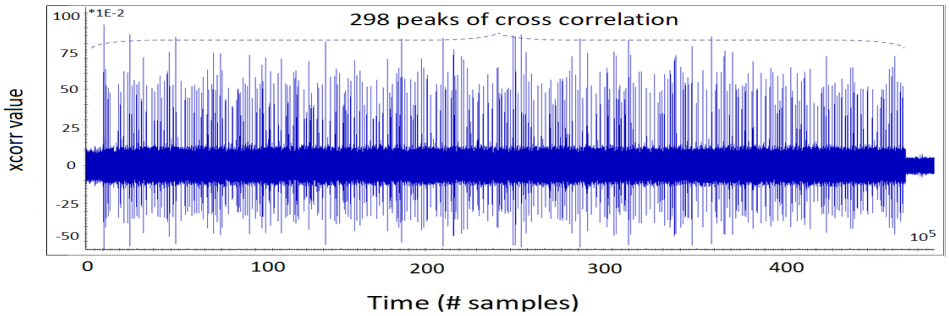

Figure (4.27) Cross-correlation between the pattern of the multiplication and the target trace

For BrainpoolP256r1, as explained in [BL], the doubling consists of 10 multiplications (except for the first doubling, where there are only 7 multiplications[10]), and the mixed addition consists of 11 multiplications.

For P-256, there is a particular parameter equal to $(-3 \mod p)$, so the multiplication by a in the doubling can be optimized. The doubling consists of 9 multiplications and the mixed addition of 11 multiplications.[11] In this way, we can "cut" the target trace in sections according to the loop of the scalar multiplication operation (as in Figure 4.28).

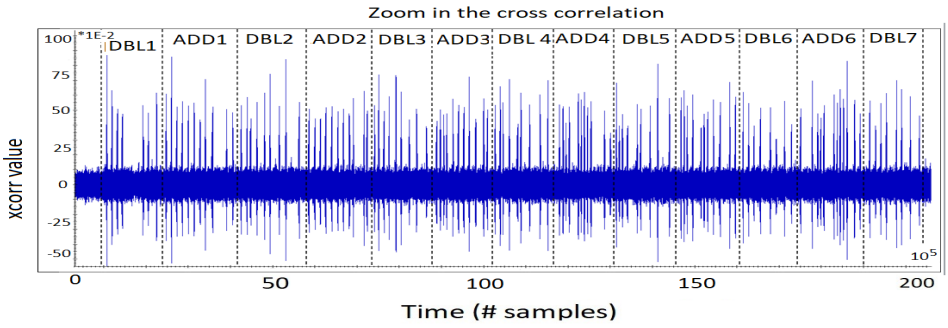

Figure (4.28) The first seven iterations of the scalar multiplication algorithm on the curve

The first iteration of the double-and-add always algorithm is completed after 18 cross-correlation peaks. For the next iterations, we take into account that each doubling consists of 9 or 10 multiplications and each addition of 11. For the first bit, the interesting section on the target is the 19th multiplication. For the second bit, the interesting section is the 39th multiplication for P-256 or the 40th multiplication for

[10] Because in the beginning $Z = 1$ and we computed aZ^4 with 3 multiplications.

[11] The fact that doubling is performed faster for P-256, allows us to recover 7 bits of the scalar at once.

BrainpoolP256r1. For the third bit, the interesting section on the target is the 59th multiplication for P-256 or the 61th multiplication for BrainpoolP256r1, and so on for all the other bits of the scalar.

As the last step of this phase, we calculate multiples of the point $\boldsymbol{P}$ using our implementation. We explain this in detail in the next section.

4.5.3.3 Template Acquisition

In mbedTLS every input point is represented in affine coordinates and then converted to Jacobian coordinates. This is a common approach for the input of constraint devices, since inversion during the doubling and adding operation using affine coordinates is not efficient.

Having the intermediate values represented in Jacobian coordinates, but the input points to the device in affine coordinates, can make the template acquisition phase complicated. Hence we need to find an input point in affine corresponding to an intermediate value in Jacobian coordinates, in order to create templates.

The *target trace* is obtained with input point $\boldsymbol{P} = (x_{\boldsymbol{P}}, y_{\boldsymbol{P}})$ given in affine coordinates. In order to compute the intermediate values of the points $2\boldsymbol{P} = (X_{2\boldsymbol{P}}, Y_{2\boldsymbol{P}}, Z_{2\boldsymbol{P}})$ and $3\boldsymbol{P} = (X_{3\boldsymbol{P}}, Y_{3\boldsymbol{P}}, Z_{3\boldsymbol{P}})$ with our implementation, we use the formulas defined in [BL]. Note that this does not correspond to the point $2\boldsymbol{P}$ and $3\boldsymbol{P}$ in affine coordinates, because $Z_{2\boldsymbol{P}}, Z_{3\boldsymbol{P}} \neq 1$. We cannot compare directly the templates with input point $2\boldsymbol{P}$ (resp. $3\boldsymbol{P}$), since they are not in affine form.

We create our templates with a specific input point $\boldsymbol{Q}_i$ such that the first field multiplication D_1 in $2\boldsymbol{P}$ or $3\boldsymbol{P}$ is the same with the one attacked on the target trace. The squaring of the X-coordinate of the intermediate value is not affected by the change of coordinates system.

The way to construct the input point for templates is more sophisticated. Let us assume that we have the input point $\boldsymbol{Q}_0 = (x_{\boldsymbol{Q}_0}, y_{\boldsymbol{Q}_0})$ in affine coordinates associated to the point value $2\boldsymbol{P}$ and $\boldsymbol{Q}_1 = (x_{\boldsymbol{Q}_1}, y_{\boldsymbol{Q}_1})$ corresponding to $3\boldsymbol{P}$. We need to analyze the squaring of the X-coordinate in Jacobian coordinates.

The input point $\boldsymbol{Q}_0 = (x_{\boldsymbol{Q}_0}, y_{\boldsymbol{Q}_0})$ should be a solution in $\mathbb{F}_p \times \mathbb{F}_p$ of the following system:

$$\begin{cases} x_{\boldsymbol{Q}_0} = X_{2\boldsymbol{P}} \\ y^2_{\boldsymbol{Q}_0} = x^3_{\boldsymbol{Q}_0} + a x_{\boldsymbol{Q}_0} + b \end{cases} \tag{4.3}$$

with a, b the parameters of the curve as defined in [BSI10, NIS13].

The number X used as input in the squaring is random, so $X^3 + aX + b$ is also random. If $x^3_{\boldsymbol{Q}_0} + a x_{\boldsymbol{Q}_0} + b$ is not a square in the finite field, we can change one bit in X as proposed in [BCP+14] to obtain another point on the curve that satisfies Equation (4.3).

We locate the first multiplication in the template trace corresponding to the squaring of the X-coordinate of the input point $\boldsymbol{Q}_0$ or $\boldsymbol{Q}_1$, depicted in Figs. 4.30, 4.31

respectively. With these two patterns and the target trace (Figure 4.29), we can perform template matching.

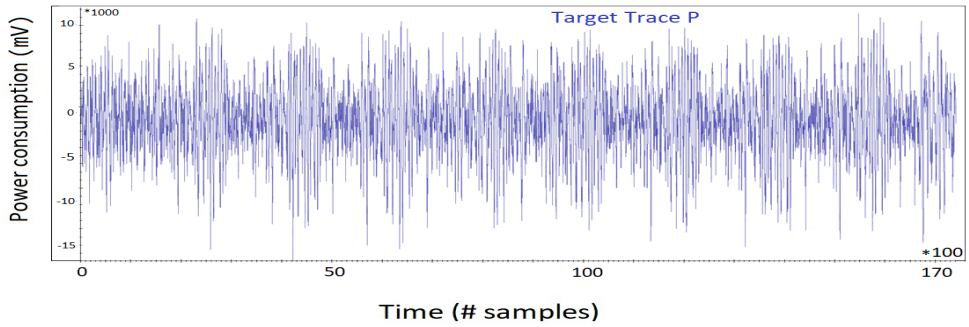

Figure (4.29) Pattern of the 19^{th} multiplication in trace with input $\boldsymbol{P}$

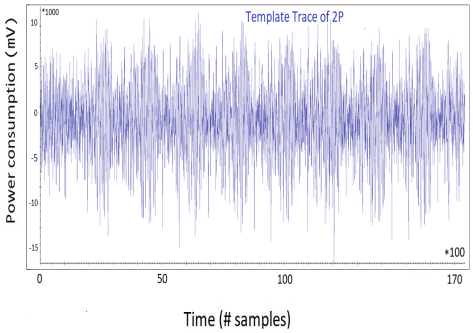

Figure (4.30) Pattern of the 1^{st} multiplication in trace with input $\boldsymbol{Q}_0$

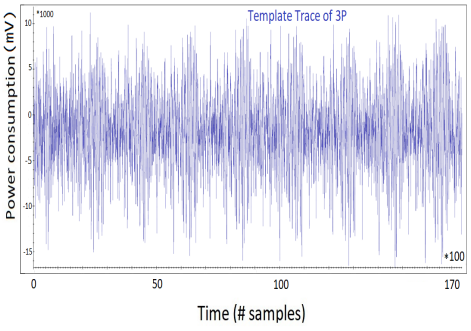

Figure (4.31) Pattern of the 1^{st} multiplication in trace with input $\boldsymbol{Q}_1$

4.5.3.4 Horizontal Leakage

We present how to perform *template matching* by making the right hypothesis on a scalar bit. This procedure is described for the cases of horizontal and vertical leakage. The probability of having horizontal leakage corresponds to the probability of having different carry propagation between the two templates.

When the traces are not synchronized (86% of cases in BrainpoolP256r1 curve, and 95% in P-256), cross-correlation between the multiplication pattern and the target trace is performed before template matching, in order to choose the correct part of the target trace. Then we align the template and target traces and decide what the correct guess for the key bit is.

Horizontal leakage is observed when there is different carry propagation between two multiplications 256×32 in the field. In Figure 4.32 we see the misalignment of the traces due to carry propagation.

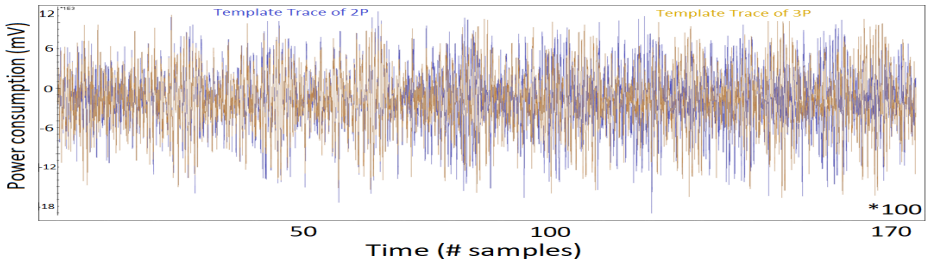

Figure (4.32) Misalignment of two template traces due to the carry propagation

4.5.3.5 Vertical Leakage

When the implementation is executed in constant time and the template traces are synchronized with the target trace, the same method as described in [BCP+14] can be used (14% of cases in brainpoolP256r1 and 5% in P-256). The carry propagation is the same between the two templates and the target as depicted in Figure 4.33; therefore, we can only observe a *vertical* leakage in our traces. In our experiments,

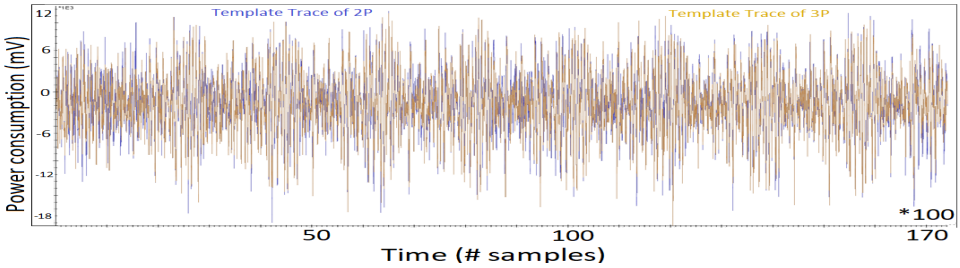

Figure (4.33) Two templates with the same carry propagation

we use the Pearson correlation coefficient $\rho(X, Y)$ as described in Chapter 3 and we get a correlation of 0.81 for the multiplication obtained from the target trace and the template trace of $2\boldsymbol{P}$. The same value drops to 0.78 for the correlation of the target trace with the template trace of $3\boldsymbol{P}$. Thus, this leads us to conclude that the second bit of the scalar is 0, which is indeed the correct key bit because the scalar used in this experiment is $\texttt{0xA5} = (10100101)_2$. However, the difference between the correct and the wrong template correlation is too small to deduce with confidence the correct bit in every round.

4.5.4 Success Rate for One Key-bit

In this part, the method used to calculate and increase the success rate of our attack is described. As explained in Section 4.5.1, the probability to have a different carry propagation between two templates is 95% for P-256 and 86% for brainpoolP256r1.

The horizontal attack scenario is easy, since if two templates have a different

carry propagation, then the success rate of finding this bit is 100%. For the vertical attack scenario, the success rate depends on the input data. Therefore, we examine only the input data, for which the carry propagation is the same in two template traces. By using random points on each curve for the target trace, we can compute the success rate in the following way:

- Acquire 30 target traces and 100 template traces for each assumption when the carry propagation is the same.
- Compute the Pearson correlation between the target and a template trace for each assumption.
- With k_i we denote the correct guess for the corresponding bit of the key k. If k_i gives the highest Pearson correlation, then the counter corresponding to the success of the attack increases. If $\neg k_i$ has higher correlation, then the counter corresponding to the failure of the attack increases.

The success rate per key bit in vertical leakage is 76.23% for P-256 and 69% for BrainpoolP256r1. The total success rate to find one key bit, independent of the leakage model, is $1\times0.95+0.76\times0.05 = 98.8\%$ for P-256, and $1\times0.86+0.69\times0.14 = 95.66\%$ for BrainpoolP256r1.

Averaging template traces can increase the vertical information leakage. When the scalar is randomized, we cannot perform the attack with more than one target trace. However, we can still acquire more than one template trace. By using only one target trace with an average of a few template traces, the success rate increases as shown the Table 4.3 on the BrainpoolP256r1 curve. For instance, by using 100 template traces the success rate for BrainpoolP256r1 curve is $1 \times 0.86 + 0.99 \times 0.14 = 99.86\%$

Table (4.3) Different success rates according to the number of average template traces on brainpoolP256r1 curve.

Number of average traces	1	10	50	100
Success Rate	69%	80.70%	91.60%	99.80%

4.5.5 Error Detection & Correction

The novelty of this method is the possibility of error detection and correction. The method of error-detection is applied when two templates have the same carry propagation. As we described in the previous section, the success rate to retrieve one bit using OTA is close to 99%, which means that there is a 1% probability of having an unsuccessful attack due to a wrong key guess.

For a 256-bit scalar, if an error occurs in the beginning and it is not detected, the success rate for the original OTA is 7.6% ($0.99^{255} \simeq 0.076$), since this error will propagate and affect all the bits after the wrong guess. Therefore, it is very important to detect and correct errors before making new templates. An error can be made when both template traces have the same carry propagation. In order to be sure for a key bit value, the value of this current bit and the following one can be computed. For instance, if the two templates for $2\boldsymbol{P}$ and $3\boldsymbol{P}$ have the same carry propagation, then we create templates for $4\boldsymbol{P}$, $5\boldsymbol{P}$, $6\boldsymbol{P}$ and $7\boldsymbol{P}$. The following four cases can occur:

1. One template has the same propagation as the target: Then, we can determine correctly both key bits. For instance, if the template trace of $7\boldsymbol{P}$ gives the same carry propagation as the target trace, then the only possible bit values are “1" for the second key bit and “1" for the third one.

2. Two templates have the same propagation as the target: We need to compute the next 4 templates ($4i, 4i+1, 4i+2, 4i+3$, where k_i is the i-th bit of the exponent), and recover the next bit using these 4 template traces.

3. Three templates have the same propagation as the target: We compute 6 templates, and recover the next bit using those 6 template traces.

4. Four templates have the same propagation as the target: We compute 8 templates, and recover the next bit using those 8 template traces.

The probability to have one template trace with the same carry propagation as the target trace on BrainpoolP256r1 is 70%, two template traces is 14%, three template traces is 3.9% , four template traces is 1.2%, five template traces is 0.4%, six template traces is 0.1%. The probability for more template traces with the same carry propagation is very low.

For P-256, the probability to have one template trace with the same carry propagation as the target trace is, 90%, two template traces is 5.9%, 3 template traces is 0.7%, 4 template traces is 0.1%, 5 template traces is 0.01%. For both curves, this probability reduces significantly for more template traces.

As a conclusion, the number of template traces cannot increase exponentially. At the end of the attack, in order to retrieve the 256-bit scalar, there can be an uncertainty for the last 2 or 3 bits. By exhaustive key search or by comparing the corresponding templates with the template of $\boldsymbol{Q} = [k]\boldsymbol{P}$, the last 2^2 or 2^3 scalar key bits can be found.

Template matching is performed at suitable parts of the traces, where key bit related assignments take place. In order to distinguish the right hypothesis on the attacked bit of the scalar, we use a pattern matching technique based on the Pearson correlation coefficient $\rho(X, Y)$ between the target trace and the template traces, as

described previously in Section 4.4 and in Chapter 3. This metric is chosen again at this point, since it is both scale and offset-shift invariant.

4.6 Classification Algorithms for Online Template Attacks

The fact that the template-building phase in OTA is not necessary, significantly simplifies the process of retrieving the key, leaving the overhead of the attack in the template-matching phase. The template-matching technique used for both OTA projects [BCP+14, DPN+16] is based on the Pearson correlation coefficient. In [OPB16], more efficient techniques from the field of Machine Learning are used as distinguishers, and the proposed attack reaches a success rate of 100% with only 20 template traces per key bit. This work is the first step towards a framework for "automating" the template-matching phase.

The attack can be classified as a form of OTA having the same attack model and assumptions. The proposed classification techniques from the field of Machine Learning (k-Nearest Neighbour, Naïve Bayes, SVM) provide an efficient and simplified way to match templates during a Template Attack with very high success rates. A practical application of this attack is demonstrated on the scalar multiplication algorithm for the Brainpool curve BP256r1 implemented in mbedTLS.

4.6.1 Machine Learning Techniques in Cryptography

Machine learning (ML) techniques are developed to provide efficient pattern recognition and feature extraction algorithms, mainly used in artificial intelligence. ML algorithms are used to build models, in order to make data-driven predictions or decisions. Applying ML techniques to cryptanalysis is not a new concept for either community. Initially proposed by Rivest in [Riv91], the use of learning algorithms can facilitate the cryptanalysis of cipher feedback systems, and data compression algorithms can be used for cryptographic purposes (for instance hash functions). Combined concepts from both fields formalized new attack techniques, such as Differential Cluster Analysis (DCA) proposed by Batina et al. in [BGLR09]. DCA uses cluster analysis to detect internal collisions and it combines features from previously known collision and DPA attack techniques. The results of fuzzy k-means clustering presented in [PITM14] using unsupervised learning for exponent recovery of a 512-bit RSA are pretty impressive, though it is quite challenging to find the points of interest that give high success rates for the attack.

Feature extraction, classification and clustering ML algorithms can be used to extract useful information from the patterns of the electromagnetic (EM) or power traces that are related to the secret key. Our proposed attack uses classification algorithms as an alternative distinguisher for the template matching phase of TA, in order to provide accurate models for distinguishing between template traces and eventually recover the key.

There are many interesting publications about the three aspects of our contribution, namely template attacks, scalar multiplication attacks and classification methods in SCA. The related work that we present here can be regarded as an intersection of any two of the previous aspects.

Lerman et al. in [LMV+13] presented a semi-supervised template attack based on the "Partitioning Around Medoids" clustering technique [TK06]. The authors show that by knowing the Hamming weight of one key byte, they are able to predict the correct key using 10 traces (each representing the average of 128 acquisitions) with 61.5% success rate. In contrast, our attack needs 20 template traces to achieve 100% success rate using three different classification methods and no knowledge of the Hamming weight is required.

Karsmakers et al. proposed data reduction techniques for the profiling phase in [KGP+09]. They showed that using principal component analysis in combination with linear discriminant analysis reduces the number of necessary time samples and therefore the computational complexity and data storage required for TAs. Our attack focuses on the template matching phase and uses the properties of classification algorithms as a distinguisher, which sorts each template in the appropriate class.

The work of this section can be considered a generalization of the attack of Hanley et al. presented in [HTM11]. In their work, the Bayes method is used in the operation level for classifying templates and distinguishing squarings from multiplications during an RSA exponentiation. We use classification for template matching of the whole scalar multiplication routine and therefore recover the scalar bits one-by-one. We also tested the validity of our attack with two more classification methods, getting the maximum success rate for all.

Heyszl et al. showed in [HIM+13] how to use unsupervised cluster algorithms to attack and recover cryptographic exponentiations. They use the k-means algorithm to combine leakage information from multiple simultaneous measurements. Although the unsupervised property of this attack is really interesting, their attack model is quite restricted. It requires simultaneous measurements with many probes and many leakage points of the device, which in real devices is very rare. Even when it happens, it would have high signal-to-noise ratio. Our attack model is generic and requires no previous knowledge of the leakage model, similar to OTA [BCP+14].

In [DPN+16] we show how to achieve high success rate in OTA and correct possible errors in recovering the scalar bits with 69.5% success rate using one average template trace. The success rate reaches 99.8% when 100 average template traces are used. The template matching that is used for both works is based on the Pearson correlation coefficient. As a follow-up of this work, we use in this section more efficient techniques from the field of ML as distinguishers, in order to achieve a success rate of 100% with only 20 template traces per key bit. We achieve higher success rates compared to related classification techniques in SCA, mainly because we manage to use very efficient distinguishers. We demonstrate the practical application of our attack on the scalar multiplication algorithm for the Brainpool curve

BP256r1 implemented in mbedTLS, as described in Section 4.5.0.1.

4.6.2 Methodology for OTA with Classification Methods

In this Section we present our attack methodology, which is very similar to OTA methodology described previously. The same attack model is assumed as in Section 4.2.1. The only difference lies in the template matching phase, where we use classification algorithms as distinguishers for templates instead of Pearson correlation. We first describe the classification algorithms we used, then the experimental setup and finally we present our results.

4.6.2.1 Classification Algorithms used for OTA

The classification algorithms that can be used in the SCA setting for ECC implementations, are described in detail in Section 3.5.3. For the template matching phase of OTA, we used the following three classification methods:

- The *Naïve Bayes Classification* is a function that combines the naive Bayes probability model with a decision rule, for instance the hypothesis that is most probable. For each class c_i, we pick as classifier the class index that gives the maximum value for an event.
- The *k-Nearest Neighbour Classification* (kNN) is a classification method based on the closest instances of the training set to the unlabeled data.
- The *Support Vector Machine (SVM)* is a supervised learning model that produces a discriminative classifier formally defined by a separating hyperplane. For our analysis, a linear SVM classifier is used.

We first run the algorithm in the target device with input point $\boldsymbol{P}$. Then, we need to find a distinguished pattern in the target trace, which we use to "build" our templates. We do that on-the-fly after the target trace is obtained. For our setup, the repetitive multiplication patterns are the same as in Section 4.5.3.3. These patterns appear for every operation for each one of the 32 bits of the scalar.

4.6.2.2 Experimental Setup

We used the traces taken from the target device used in [DPN+16], as described in Section 4.5.3.1. The target and template traces are obtained from the same device, i.e. the STM32F4 platform with an ARM Cortex-M4 processor running at 168 MHz.

The target implementation in mbedTLS accepts as input a point $\boldsymbol{P}$ on an elliptic curve in affine coordinates and the scalar $k = (k_0, k_1, ..., k_{l-1})$. The auxiliary point $\boldsymbol{Q}$ is used to store the result of every round and its value is returned at the end of the algorithm. There is another point $\boldsymbol{Q}_1$, which is used to store the dummy addition.

For every bit of the scalar the doubling of point $\boldsymbol{Q}$ is calculated and then addition with $\boldsymbol{P}$ is performed. According to the scalar bit, the result of the addition is stored either in $\boldsymbol{Q}$ or in $\boldsymbol{Q}_1$, but the value of $\boldsymbol{Q}$ is always used for the next round. In that way, the algorithm is regular and runs in constant time. The change of coordinates from affine to Jacobian and then back to affine is done for efficiency reasons (addition with mixed coordinates is faster than addition with affine coordinates).

Algorithm 11: Double-Add-Always mixed coordinates

Input: $\boldsymbol{P} = (x, y)$ and scalar $k = (k_0, k_1, ..., k_{l-1})$
Output: $\boldsymbol{Q} = [k]\boldsymbol{P}$

1: **if** $\boldsymbol{P} \in \mathcal{E}$ and $k \in \mathbb{Z}$ **then**
2: $\quad \boldsymbol{Q} \leftarrow 0, \boldsymbol{Q}_1 \leftarrow 0, l \leftarrow \text{len}(k)$
3: $\quad$ **for** i from $l-1$ down to 0 **do**
4: $\quad\quad \boldsymbol{Q} \leftarrow DBL(\boldsymbol{Q})$
5: $\quad\quad$ **if** $k = 1$ **then**
6: $\quad\quad\quad Q \leftarrow ADD_{\text{mixed\_add}}(Q, P)$
7: $\quad\quad$ **else**
8: $\quad\quad\quad Q_1 \leftarrow ADD_{\text{mixed\_add}}(Q, P)$
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: $\quad \boldsymbol{Q} : (x, y) \leftarrow (X/Z^2, Y/Z^3)$
12: $\quad$ return Q
13: **else**
14: $\quad$ **error**
15: **end if**

For the template matching phase we used a computer running 64-bit Windows 7 with an Intel processor i5-4590 CPU at 3.3 GHz. The software implementations are written in Matlab (version R2014a) environment. The classification methods are implemented using the Matlab functions `fitcsvm` for SVMs, `fitcknn` for k-Nearest Neighbour and `fitNaiveBayes` for Naive Bayes Classification. Those functions use as training data the template traces and the multiplication pattern and return the correct classification of that pattern. For all the methods, we used the 1000 samples long pattern from the target trace ($sample = target\_trace(1 : 1000)$) as the sample that we want to classify. For the SVM classifier, we used the default Gaussian kernel provided in the software with two-class learning. The `fitNaiveBayes(X,Y)` function returns a Naive Bayes classifier model trained by predictors X and class labels Y.

For our experiments, the template traces for each input point are the predictors and the actual point that they represent works as the class label. In later versions of Matlab (starting from R2015b) this function is replaced by the `fitcnb` method,

which works in a similar way but offers additional options, such as specifying a distribution to model the data or prior probabilities assigned to classes. We trained a 4-nearest neighbor classifier with the following properties `fitcknn(templates, class_number, 'NumNeighbors', 4)` and the default Euclidean distance is chosen as metric.

4.6.3 Classification Scores for each Method

From the specifications of the BP256r1 curve [BSI10], we know that the first iteration of the algorithm consists of 18 multiplications and the remaining iterations need 21 multiplications. Therefore, we cross-correlate the multiplication pattern with the target trace and look for the 19th multiplication, where the manipulation of the point $2\boldsymbol{P}$ starts.

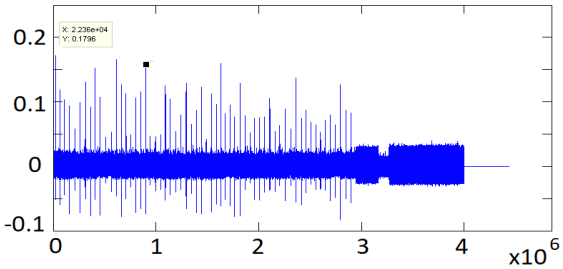

Figure (4.34) Cross-correlation of multiplication pattern with the target trace

Since we are able to run the algorithm with input points of our choice for acquiring template traces, we choose the multiples $2\boldsymbol{P}$ and $3\boldsymbol{P}$ of our initial input point $\boldsymbol{P}$. If the attacker obtains the template traces in a similar device as the target, then the noise levels of the signal will be different; obtaining 10 templates for each input point and taking the average trace cancels out those differences. When the template traces for $2\boldsymbol{P}$ and $3\boldsymbol{P}$ are obtained, we cross-correlate the multiplication pattern with the template traces and we focus in the beginning of those graphs. As it can be observed in Figures 4.34, 4.35, 4.36 the result on cross-correlation looks almost the same for both templates. However, the classification algorithms that we used, gave as output the correct template (corresponding to the correct bit of the scalar) every time we ran the experiments. We obtained 10, 20, 40, 80 and 160 template traces for each point. For each classification algorithm the template traces were used as training data, having a class "2" for data corresponding to the template traces of $2\boldsymbol{P}$ and a class "3" for the templates traces of $3\boldsymbol{P}$. Then we used the template of the multiplication obtained from the target trace as input to the classification algorithms. According to the result of the classification, we could decide on the second most significant bit. In our case, we used the scalar $k = 10100101$ and we got the

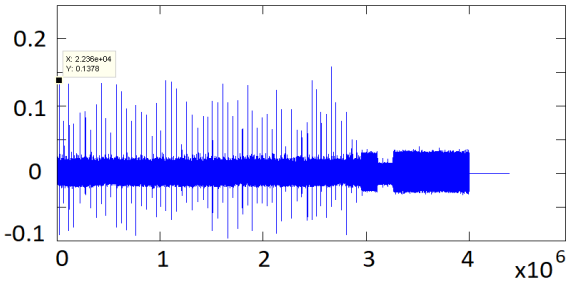

Figure (4.35) Cross-correlation of multiplication pattern with template trace $2\boldsymbol{P}$

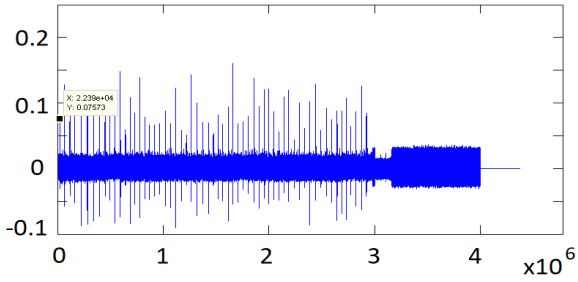

Figure (4.36) Cross-correlation of multiplication pattern with template trace $3\boldsymbol{P}$

correct result, namely classification of the multiplication pattern in class "2", for all the algorithms used.

The procedure described in this section can be followed iteratively to recover all the bits of the scalar.

The proposed attack has 100% success rate for each classification method used. We can obtain this result with one template trace for $2\boldsymbol{P}$ and one for $3\boldsymbol{P}$, but in order to create training data sets and achieve high classification scores, we used more templates. It is shown in Section 4.4 that Naïve Bayes and kNN methods give always the correct classification with score [1,0]. The entries of this table are in the form “a [b, c]”, where the classification output is indicated as "a" (class that the chosen interval of the target trace belongs to) and [b, c] is the posterior probability for the given classification method (1: all the training data belong in the first class, 0: none of the training data were classified as elements of the second class).

However, for the SVM method, we notice that with fewer templates, the training data are closer to the hyperplane. The notation [a,-a] for the SVM method indicates the distance of the data set from the hyperplane. Therefore, in cases where the signal is more noisy, it is recommended to use more than 20 template traces for the SVM training data, in order to have more accurate results.

Table (4.4) Different success rates according to the number of average template traces on BP curve.

♯Templates	Naïve Bayes	kNN ($k = 4$)	SVM
160	2[1,0]	2[1,0]	2[1.084593, -1.084593]
80	2[1,0]	2[1,0]	2[1.040720, -1.040720]
40	2[1,0]	2[1,0]	2[0.675875, -0.675875]
20	2[1,0]	2[1,0]	2[0.554645, -0.554645]

These success rates can be achieved only under the conditions described in Section 4.2.2. Implementing randomization of the input point to the algorithm, changes the interesting pattern of the multiplication and therefore, it cannot be classified correctly to one of each class.

4.7 Conclusions

At this point, it is clear that OTA and the adaptive template attack techniques of Sections 4.5, 4.6 are very efficient methods to attack scalar multiplication during the execution of ECC protocols. These methods can be easily adapted to other scalar multiplication algorithms as described in [Joy03, Riv11]. For binary algorithms, such as Montgomery Ladder [JY02], we can create templates for the doubling operation and find the correct key bit. For non-binary algorithms using windows, we can obtain templates for all hypotheses and perform the same attack with more template traces.

Error detection and correction of a wrong key bit guess is possible for the adaptive template attack, and it increases the success rate of the attack from 7.6% to 99.8% for a 256-bit scalar. We achieve these results by averaging 100 template traces and using two template traces to recover each key bit, instead of using one template trace as initially proposed.

Since the most commonly used scalar multiplication algorithms are vulnerable to OTA, it is interesting to see which countermeasures can be applied against it. We hereby give a list of the classical countermeasures against side-channel attacks and their efficiency against OTA:

- *Randomization of the scalar.*
  Randomizing the scalar results in getting traces with $[k']P$ instead of $[k]P$, with k' defined below. The important property that thwarts scalar randomization, is the fact that we need only one target trace, which is taken using the same randomized k' that is manipulated throughout the attack. For the template traces, we always need the first part of the trace, which corresponds to the beginning of the scalar multiplication algorithm running with input point

a multiple m of P. This part of the trace is not affected by the scalar randomization. The different ways of scalar randomization are:

1. $[k]\boldsymbol{P} = [k-r]\boldsymbol{P} + [r]\boldsymbol{P}$, two scalar multiplications are computed $\boldsymbol{Q} = [k-r]\boldsymbol{P}$ and $\boldsymbol{R} = [r]\boldsymbol{P}$;
2. $[k]\boldsymbol{P} = [k \times r^{-1}]([r]\boldsymbol{P})$, two scalar multiplications are computed $\boldsymbol{Q} = [r]\boldsymbol{P}$ and $\boldsymbol{R} = [k \times r^{-1}]\boldsymbol{Q}$;
3. $[k]\boldsymbol{P} = [k \mod r]\boldsymbol{P} + [\lfloor \frac{k}{r} \rfloor]([r]\boldsymbol{P})$, three scalar multiplications are computed $\boldsymbol{Q} = [k \mod r]\boldsymbol{P}$, $\boldsymbol{R} = [r]\boldsymbol{P}$ and $\boldsymbol{S} = [\lfloor \frac{k}{r} \rfloor]\boldsymbol{R}$.
4. $[k]\boldsymbol{P} = [k']\boldsymbol{P} - [rq]\boldsymbol{P}$, where q is the order of the curve and $k' = k + rq$ is the randomized scalar. Thus, $[rq]\boldsymbol{P} = \mathcal{O}$ the neutral element on $\mathcal{E}$.

For all these randomization techniques, OTA can be applied; the *target trace* requires one acquisition on the second or third scalar multiplication. This acquisition is possible using an oscilloscope with big memory. For the template traces, we make assumptions for each part of the scalar multiplication. We retrieve a random scalar part for each scalar multiplication part. In order to retrieve the scalar, we compute the addition (randomization 1.), the multiplication (randomization 2.) or both addition and multiplication of the scalar (randomization 3.). For the fourth case, we can recover the randomized scalar k'. Therefore, scalar randomization is not efficient against our type of attack.

- *Randomization of the point.*
  The Jacobian representation of the point can be easily randomized similar to projective coordinates. For Jacobian coordinates, the randomization consists of selecting a random r in the finite field $\mathbb{F}_p$, and computing: $(X, Y, Z) \mapsto (r^2 \times X, r^3 \times Y, r \times Z)$. In most cases, the input point is in affine coordinates, so the randomization of the point is reduced to compute: $(x, y) \mapsto (r^2 \times x, r^3 \times y, r)$. The supplementary cost of this countermeasure is 5 finite field multiplications. For comparison, the cost of one scalar multiplication using 256-bit scalar with a regular algorithm such as double-and-add-always is 5100 multiplications. Applying randomization of the input point does not allow to predict intermediate values of the calculation and prevents the construction of the template traces in a deterministic way. This countermeasure is implemented in mbedTLS, and it should be used when the device under attack has a random generator.

- *Random isomorphic elliptic curve*
  The idea to protect scalar multiplication by transforming a curve through various random morphisms was initially proposed by Joye and Tymen in [JT01b]. Assume that ϕ is a random isomorphism from $\mathcal{E}_K \rightarrow \mathcal{E'}_K$, which maps $\boldsymbol{P} \in \mathcal{E}_K \rightarrow \boldsymbol{P'} \in \mathcal{E'}_K$. Multiplying $\boldsymbol{P'}$ with k will give $\boldsymbol{Q'} = [k]\boldsymbol{P'} \in \mathcal{E'}_K$.

With the inverse map ϕ^{-1} we can get back to $\boldsymbol{Q} = [k]\boldsymbol{P}$. Our attack requires knowledge of the internal representation of the point, so if $\boldsymbol{P'}$ is on a different curve that the adversary does not know, he cannot create input points in this representation.

Horizontal leakage due to conditional statements was not expected to be seen in recent cryptographic implementations, but unfortunately they are still used. An implementation without conditional statements would not prevent the adaptive template attack, but it would reduce its success rate to that of the original OTA.

Randomizing the input point, by randomizing its coordinates, for every execution of the attacked algorithm is the most efficient countermeasure against OTA, though incurring with some cost for the performance of the implementation. Point randomization is efficient against our attack, since we need to know the input point and its intermediate values, in order to produce template traces. Actually, the adversary needs to be able to choose input points for templates. The countermeasure of point blinding must be activated, in order to use mbedTLS in ARM embedded devices. Adoption of this countermeasure is not straight-forward, because it requires the use of a random generator in the device under attack.

This chapter presented also a new approach to template matching by using classification algorithms as distinguishers. We showed the applicability of classification algorithms in an OTA scenario, where we successfully recovered bits of the scalar. The classification methods naïve Bayes, kNN and SVM can give 100% success rate in classifying correctly template traces, when the templates are chosen and built correctly. The fact that our attack scenario assumes binary classification, makes the results really accurate and can explain the absolute success rate.

Chapter 5

Boolean Exponent Splitting

> Το όλον είναι μεγαλύτερο από το άθροισμα των μερών του - The whole is more than the sum of its parts.
>
> *Aristotle*

A typical countermeasure against side-channel attacks consists of masking the intermediate values with a random number. In symmetric cryptographic algorithms Boolean shares of the secret are typically used, whether in asymmetric algorithms the secret exponent/scalar is masked using group arithmetic properties. This chapter presents a new exponent splitting technique with minimal impact on performance based on Boolean shares. More precisely, it is shown how an exponent can be efficiently split into two shares, where the exponent is the XOR sum of the two shares, typically requiring only an extra register and a few register copies per bit. Our novel exponentiation and scalar multiplication algorithms can be randomized for every execution and combined with other blinding techniques, maintaining at the same time the regularity feature. In this way, both the exponent and the intermediate values can be protected against various types of side-channel attacks. We perform a security evaluation of our algorithms using simulations based on the Mutual Information framework and we verify formally that they are secure against first-order attacks. The resistance of the proposed algorithms against side-channel attacks is practically verified with test vector leakage assessment performed on a Xilinx's Zynq zc702 evaluation board.[1]

[1] The coauthors of the corresponding paper created the diagrams of MI and performed the practical TVLA experiments; both are included at the end of this chapter for the sake of completeness.

5.1 Introduction

In public key cryptography one typically uses countermeasures based on redundant representations to prevent side-channel leakage [WMPW98, Cor99] (referred to as blinding). Exponentiation algorithms in cryptographic groups are common targets of side-channel attacks, and they tend to be prohibitively expensive to protect using provable countermeasures like masking. To protect an exponent used in a group exponentiation one would typically add a random multiple of the order of the group to the exponent, providing a random bitwise representation of the exponent. These countermeasures can provide a strong resistance to differential power analysis (DPA), but are not convenient in some instances. As noted by Smart et al. [SOP08], the random value used to blind an exponent needs to have a bit length larger than the longest run of zeros or ones in the bitwise representation of the order of the group. If we consider ECDSA [Nat09], for example, the bitwise representations of the orders of the groups used typically contain long runs of ones making this countermeasure undesirable. That is, to provide a randomized bitwise representation the random value used would have a bit length comparable to the exponent it is protecting, leading to a prohibitive impact on performance. While some have proposed alternative elliptic curves that avoid this problem [LMSS14], widely used curves will continue to have long runs of zeros and ones in the bitwise representation of the order of the group, since the modulus is typically chosen with this property to allow for the rapid computation of a modular reduction. See, for example, the elliptic curves proposed by Bernstein [Ber06], Hamburg [Ham15] and Bos et al. [BCLN16].

An attack applicable to exponents blinded by adding a random multiple of the group order has been proposed by Walter [Wal01], where one seeks to determine if the input to one operation in an exponentiation is the same as the input to another operation. This is achieved by comparing traces of the instantaneous power consumption during consecutive group operations. Many analyses of how this attack could be applied to cryptographic algorithms have been described in the literature [OS02, AF08, KKYH10, WvWM11, CFG+12, BJ13, BJPW14]. Hanley et al. [HKT15] have also shown that these attacks could be applied to a single trace. Hence, protecting an exponent by adding a random multiple of the order of the group may not be sufficient, since the randomized exponent is still equivalent to the exponent. Furthermore, Hanley et al. further demonstrated that one can construct an attack by determining if the output of one operation in an exponentiation is the same as the input to another operation. These attack techniques can also be applied to scalar multiplication algorithms for Elliptic Curve Cryptography (ECC), for instance when additive scalar blinding is used, as shown in [FRV14]. Furthermore, online template attacks, as presented in Chapter 4, make it possible to attack implementations of a blinded scalar by using one trace from the device under attack and several template traces from a similar device running the same implementation. It is, therefore, clear that the various ways of exponent (or scalar) blinding do not

provide a single, efficient countermeasure against recent attacks.

5.1.1 Related Work

Exponent splitting is a known countermeasure that comes in various forms (additive, multiplicative, Euclidean) [CJ01, CJ03]. It is generally avoided in practice, because it typically doubles the execution time. Negre and Plantard [NP16] proposed a regular modular exponentiation using multiplicative half-size splitting. This method, although performing 16% faster than related splitting algorithms, is still not very efficient since two exponentiations (of half-sized words) have to be performed. Moreover, it is only secure against SPA, but not against DPA, collision or template attacks. The authors propose randomizing the exponent to protect against those types of attacks, which should have additional performance overhead.

The countermeasures proposed in this chapter are based on a novel exponent splitting technique with minimal performance overhead. Apart from randomizing the exponent, the algorithms can be used to hide the exponent length and prevent leakage from intermediate values. One could also consider our algorithms as an enhancement of algorithmic countermeasures to address-bit side-channel attacks (ADPA) [MDS99, IIT02], where one attempts to conduct a DPA on the addresses of registers or memory locations rather than the intermediate states of an algorithm.

May et al. [MMS01] proposed a hardware countermeasure to address-bit side-channel attacks where registers used to compute an exponentiation algorithm would be randomly renamed. This idea was extended to software by Itoh et al. [IIT03] and Izumi et al. [IISO10] who proposed exponentiation algorithms where accesses to memory locations, or registers, is randomized from one execution to another.

Address-bit side-channel attacks, and the specific countermeasures, have received relatively little attention in the literature since blinding methods typically randomize the bitwise representation of an exponent, and therefore the accessed addresses [Cor99, CJ01].

5.1.2 Our Contribution

In this chapter, we present a *new countermeasure for exponent splitting*. We describe a method of splitting an exponent into *two Boolean shares*, analogous to the countermeasures that one would use for an implementation of a block cipher and similar to the countermeasures used to prevent address-bit side-channel attacks [MDS99, IIT02]. Having embedded devices as our targeted implementation and an adversary able to get useful information from the length of the exponent or the intermediate values, we provide a number of algorithms that resist a broad range of side-channel attacks.

At the same time, the modifications that are required to a group exponentiation algorithm have negligible effect on the time required to compute the actual group

exponentiation, which is a significant advantage over previous examples of exponent splitting [CJ01, CJ03].

In addition, our method can be efficiently combined with blinding techniques of the message or the point on a curve, in order to prevent leakage of the intermediate values. The blinded versions of our main algorithm come at the cost of only an inversion and a multiplication (or subtraction and addition in the case of elliptic curves). Therefore, this combined countermeasure has minimal cost in performance. We further demonstrate that the proposed algorithms can be used to compute an ECDSA signature where the nonce is generated in two shares that do not need to be combined. This approach offers a convenient method for implementing a side-channel resistant instance of ECDSA with limited significant impact on performance.

A further property that two of our algorithms have is that the exponent (or scalar) length can be hidden. The way the operations are handled by registers combining the Boolean splitting of the exponent can provide some algorithms with the useful property of tolerating leading zero bits.

Finally, the method of Boolean exponent splitting, cornerstone of the proposed algorithms, is evaluated in terms of security. An evaluation using the information-theoretic framework of Standaert et al. [SMY09] and a Test Vector Leakage Assessment (TVLA) by Goodwill et al. [GJJR11] are performed. We investigate the usual leakage models based on data or location leakage and show that an adversary would need either a second-order data attack or a third-order location attack to successfully break the security of our algorithms. In addition, we present for the first time a hybrid model, where data leakage is combined with location leakage in a 3D graph, offering new exploitation opportunities. The rich interactions between data and location leakage corroborates the need for holistic countermeasures that encompass a wide spectrum of side-channel attacks.

5.1.3 Organization of this Chapter

This chapter presents efficient algorithmic countermeasures that can be applied to both exponentiation and scalar multiplication algorithms. More precisely, in Section 5.2 the necessary preliminary notions are presented. In Section 5.2.1, we describe previously proposed exponent splitting methods, their efficiency and known attacks against these countermeasures, and in 5.2.2 some prerequisites and terminology for provably secure algorithms. Section 5.3 presents our proposed methods of exponent splitting based on an XOR operation, applied on various regular algorithms. A discussion on vulnerabilities in different versions of previously proposed regular algorithms, is followed by our proposed modification, in order to achieve a secure, highly regular and randomized exponentiation algorithm. In Section 5.3.4 we present our splitting method in the context of scalar multiplication for elliptic curves. In Section 5.4, we demonstrate how this method can be used to generate an ECDSA signature where only shares of the random nonce are manipulated. Sec-

tion 5.5 presents the security evaluation of our algorithms based on formal methods and the information theoretic framework. In Section 5.5.5 implementation considerations are discussed and the results of TVLA of a practical implementation on a Xilinx Zynq evaluation board are presented. Finally, we conclude in Section 5.6.

5.2 Preliminaries

5.2.1 Exponent Splitting Methods

The critical operation in public key cryptographic algorithms is exponentiation in a given group $\mathbb{G}$ of order $|\mathbb{G}| = m$, where the input message $x \in \mathbb{G}$ is raised to a secret exponent κ and the result $y = x^{\kappa}$ is the public output of the algorithm. When implementing a group exponentiation algorithm the exponent is typically blinded by adding some random multiple of the group order to the exponent. Trivially, $(rm) + \kappa \equiv \kappa \pmod{m}$ for $r, \kappa \in \mathbb{Z}$ where r is random. Hence, computing $x^{\kappa+rm}$ is equivalent to computing x^{κ}. While this randomizes the bitwise representation of an exponent, the entire exponent is still equivalent to the exponent in a given group. For scalar multiplication over certain cryptographic elliptic curves, typically those over a prime field $\mathbb{F}_p$ where p is a (generalized) Mersenne number, adding a random multiple of the group order does not hide all the bits of scalar k [CJ03]. Examples of attacks that have been proposed include analyzing a single trace (from SPA [KJJ99] to collisions in manipulated values [KKYH10,WvWM11,HKT15]) or attempting to find collisions in the random values used to then derive a (blinded) exponent [SI11].

One method that can hinder these attacks, is to split an exponent into two values whose bitwise representations are random. Then one would compute a group exponentiation where the combined effect of the two values is equivalent to that of the desired exponent. There are several methods of exponent splitting proposed by Clavier and Joye [CJ01]:

- **Additive Splitting.** For a random integer r with bit-length larger than or equal to the exponent k, we can define $k = r + (k - r)$. The output of the modular exponentiation $y = x^k$ in $\mathbb{G}$ can be computed by calculating $y = x^r \cdot x^{k-r}$ in $\mathbb{G}$.

- **Multiplicative Splitting.** For some group $\mathbb{G}$ we can define $k' = k\,r^{-1} \bmod |\mathbb{G}|$ for some integer r, which is relatively prime to the group order. Then the exponentiation $y = x^k$ in $\mathbb{G}$ can be computed by using $y = (x^r)^{k'} \bmod |\mathbb{G}|$.

The same techniques can be applied to scalar multiplication algorithms for elliptic curves (ECs), in order to hide the secret scalar. The problem with these methods of exponent splitting is that they will typically double the time required to compute a group exponentiation, because r is required to have a similar bit-length to the

exponent. A practical attack by Feix et al. [FRV14] demonstrates that a blinded scalar can be determined if r is too small.

A further method described by Ciet and Joye [CJ03] is:

- **Euclidean Splitting.** By writing the exponent as $k = \lfloor k/r \rfloor r + k \bmod r$ and letting $s = x^r$ for some r, then $y = x^k$ can be computed by $y = s^{k'} \times x^{k \bmod r} = (x^r)^{k'} \times x^{k \bmod r}$, where $k' = \lfloor k/r \rfloor$.

The impact on the time required to compute an exponentiation is lower than the other splitting methods listed above. In [CJ03] it is concluded that this variant applied to Shamir's double ladder has the same cost as the 'double-and-add-always' algorithm (equivalent to the 'square-and-multiply-always' for exponentiation). Precomputation of powers of s can reduce the exponentiation cost compared to additive or multiplicative splitting. However, this method has the same constraints as adding a multiple of the group order. That is, r needs to have a bit length larger than the longest run zeros in k; otherwise, a big part of the exponent will appear in clear [CJ03]. This fact will have a significant impact on performance in many cases [SOP08].

5.2.2 Provable Security

This Section provides a simplified explanation of some basic definitions regarding *non-interference (NI)* that are useful for the rest of this chapter. The interested reader can refer to the paper of Ishai, Sahai and Wagner [ISW03] for the initial definitions of private circuits and probing attacks and to the paper of Barthe et al. [BBD+16] for the explanation of strong non-interference in higher order masking.

The baseline notion of security for masked algorithms is *t-probing security*. In the current state-of-the-art, masking schemes usually come with a security proof in the so-called *probing model* [ISW03], which offers a realistic framework in the theoretical evaluation of implementations that offer security against side-channel attacks. In this model, an adversary can perform a probing attack by placing a metal needle on a wire of interest and reading off the value carried along that wire during the target's computation.

A basic notion in probing models is the *private circuit*, which is a Boolean circuit as defined in [ISW03], whose vertices are (Boolean) gates and whose edges are wires. A randomized circuit is a circuit augmented with random gates, namely gates that produce a random output, uniformly and independently of everything else and fresh for each invocation of the circuit. Gadgets are used to implement the functionality of private circuits, i.e. using a probabilistic function $f : \mathbb{F}_q^n \to \mathbb{F}_q^{m'}$ on a n-bit input, they produce a uniformly random m-bit output. According to [CS18], the notion of t-NI security of gadgets is defined as follows:

Definition 5.2.1. A gadget is t-non-interfering (t-NI) if and only if every set of at most t internal probes can be simulated with at most t shares of each input.

For instance, a gadget is 2-NI or 2-probing secure, if the joint distribution of any set of at most 2 of its positions, corresponding to adversary probes, depends on at most 2 shares of the gadget's inputs. This guarantees that, if the input is uniform, no information about it leaks through any 2 probes in the circuit. Extending this definition to the algorithm level, an algorithm $\mathcal{A}$ is $t-$probing secure, if the observation of up to t intermediate computations in the implementation during the execution of $\mathcal{A}$ does not reveal anything about the secret variables (held by its inputs) [BBD+16].

In the following sections, we define some alternative algorithms for splitting exponents and scalars using an XOR operation. As this is a sort of masking, we also provide proofs using the t-NI notion introduced here and we show that implementations of these algorithms are secure against first-order side-channel analysis.

5.3 Boolean Exponent Splitting Methods

In this section, we propose methods of exponent splitting based on XOR operation, and how an XOR-split exponent can be applied to the Montgomery ladder.

5.3.1 Montgomery Power Ladder

The algorithm for the Montgomery Power Ladder (MPL) is described in Chapter 2. We recall the description of the MPL given by Joye and Yen [JY02]: We consider the problem of computing $y = x^{\kappa}$ in the multiplicative group $\mathbb{G}$ for inputs x and κ. Let $\sum_{i=0}^{n-1} k_i\, 2^i$ be the binary expansion of κ with bit length n (for ease of expression we shall also denote this as $(k_{n-1}, \ldots, k_0)_2$ where convenient). Then, defining $L_j = \sum_{i=j}^{n-1} k_i\, 2^{i-j}$ and $H_j = L_j + 1$, we have

$$L_j = 2\, L_{j+1} + k_j = L_{j+1} + H_{j+1} + k_j - 1 = 2\, H_{j+1} + k_j - 2 \tag{5.1}$$

and so we obtain

$$(L_j, H_j) = \begin{cases} (2\, L_{j+1},\, L_{j+1} + H_{j+1}) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1},\, 2\, H_{j+1}) & \text{if } k_j = 1\,. \end{cases} \tag{5.2}$$

If we consider one register containing x^{L_j} and another containing x^{H_j} then (5.2) implies that

$$(x^{L_j}, x^{H_j}) = \begin{cases} \left(\left(x^{L_{j+1}}\right)^2,\, x^{L_{j+1}} \cdot x^{H_{j+1}} \right) & \text{if } k_j = 0\,, \\ \left(x^{L_{j+1}} \cdot x^{H_{j+1}},\, \left(x^{H_{j+1}}\right)^2 \right) & \text{if } k_j = 1\,. \end{cases} \tag{5.3}$$

Given that $L_0 = k$ one can build an exponentiation algorithm that requires two group operations per bit of the exponent. Joye and Yen give several different versions, one of which is shown in Algorithm 12; these algorithms are all highly regular, meaning that a deterministic sequence of operations is executed for an exponent

of a given bit length. Here $1_{\mathbb{G}}$ represents the neutral element of the multiplicative group $\mathbb{G}$.

Algorithm 12: Montgomery Ladder

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = \sum_{i=0}^{n-1} k_i\, 2^i$
Output: x^κ

$R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow x$;
for $i = n-1$ **down to** 0 **do**
  $R_{\neg k_i} \leftarrow R_{k_i} \cdot R_{\neg k_i}$;
  $R_{k_i} \leftarrow (R_{k_i})^2$;
end
return R_0

In applying an XOR-split exponent to MPL we use one share to dictate the address accessed and the other to act as the exponent. That is, we consider (5.2), where the previous round may provide either (L_j, H_j) or (H_j, L_j) and the computation is changed accordingly.

Let $S_{0,j} = L_j$ and $S_{1,j} = H_j$ and $\sum_{i=0}^{n-1} a_i\, 2^i$ be the binary expansion of A with bit length n (i.e. the same bit length as the exponent). Then we can use the values of a_i to dictate whether a pair of registers holds (L_j, H_j) or (H_j, L_j). Specifically, (5.2) can be rewritten as

$$(S_{a_j,j}, S_{\neg a_j,j}) = \begin{cases} (2\, S_{a_j,j+1},\, S_{a_j,j+1} + S_{\neg a_j,j+1}) & \text{if } k_j = 0\,, \\ (S_{a_j,j+1} + S_{\neg a_j,j+1},\, 2\, S_{\neg a_j,j+1}) & \text{if } k_j = 1\,. \end{cases} \tag{5.4}$$

In (5.4), the values of L_j and H_j are assigned to S in an order dictated by the binary expansion of A. Generating A as a random sequence of bits could provide some side-channel resistance, but does not protect the exponent.

We further consider $\sum_{i=0}^{n-1} a_i\, 2^i$ and $\sum_{i=0}^{n-1} b_i\, 2^i$ be the binary expansion of A and B, respectively, where $\kappa = A \oplus B$ of bit length n. We note that, as above, $\sum_{i=0}^{n-1} k_i\, 2^i$ is the binary expansion of κ and $k_i = a_i \oplus b_i$ for $0 \le i < n$. Then (5.4) can be rewritten as

$$(S_{a_j,j}, S_{\neg a_j,j}) = \begin{cases} (2\, S_{b_j,j+1},\, S_{b_j,j+1} + S_{\neg b_j,j+1}) & \text{if } k_j = 0\,, \\ (S_{b_j,j+1} + S_{\neg b_j,j+1},\, 2\, S_{\neg b_j,j+1}) & \text{if } k_j = 1\,. \end{cases} \tag{5.5}$$

Rather than using the same value to control which order L_j and H_j are assigned and read, we use the bits of A to determine the order in which L_j and H_j are assigned, and the bits of B to determine the order they are read. The combined effect is that the order the L_j and H_j are assigned and read is dictated by the bits of κ.

We note that (5.5) implies

$$(x^{S_{a_j,j}}, x^{S_{\neg a_j,j}}) = \begin{cases} \left(\left(x^{S_{b_j,j+1}}\right)^2, x^{S_{b_j,j+1}} \cdot x^{S_{\neg b_j,j+1}}\right) & \text{if } k_j = 0\,, \\ \left(x^{S_{b_j,j+1}} \cdot x^{S_{\neg b_j,j+1}}, \left(x^{S_{\neg b_j,j+1}}\right)^2\right) & \text{if } k_j = 1\,. \end{cases}$$

Next we present Algorithm 13, which operates in much the same way as the MPL, as it produces a regular sequence of multiplications and squaring operations. However, one more register is required to allow the assignment in line 5 to affect R_0 or R_1. This algorithm presents the core idea of our Boolean-split exponent method. Extra security features, such as hiding intermediate values, can be provided by combining countermeasures, for instance applying blinding of the input point. Algorithm 13 is largely equivalent to an algorithm proposed by Izumi et al. [IISO10] were we set the multiplication in line 4 to operate in a random order as it provides a better resistance to collision attacks, as demonstrated by Kim et al. [KKYH10]. We discuss this further in Section 5.3.4.

Remark 1. Algorithm 13 and the version proposed by Izumi et al. [IISO10] would also potentially leak one bit of the exponent used if the Hamming distance model is considered [MOP07]. That is, after the most-significant bit of the exponent is treated R_0 and R_1 contain x and x^2. When the next most-significant bit is set to one they will be overwritten by x^3 and x^4, whereas if the next most-significant bit is set to zero they will be overwritten by x^2 and x^3. In the latter case x^2 may overwrite x^2 producing a Hamming distance of zero. Depending on the microprocessor, the difference between zero and a random Hamming distance could be quite significant. Hence, Algorithm 13, and likewise the algorithm proposed by Izumi et al. [IISO10], should not be used where leaking the second-most significant bit could compromise the security of an implementation.

Algorithm 13: Montgomery Ladder with XOR-Split Exponent I

Input: $x \in \mathbb{G}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$
Output: x^κ where $\kappa = A \oplus B$

1 $R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow 1_{\mathbb{G}}$; $R_2 \leftarrow 1_{\mathbb{G}}$;
2 $b' \xleftarrow{R} \{0,1\}$; $R_{\neg b'} \leftarrow x$;
3 **for** $i = n-1$ **down to** 0 **do**
4 $\quad R_2 \leftarrow R_{a_i} \cdot R_{\neg a_i}$;
5 $\quad R_{a_i} \leftarrow \left(R_{(b_i \oplus b') \oplus a_i}\right)^2$;
6 $\quad R_{\neg a_i} \leftarrow R_2$;
7 $\quad b' \leftarrow b_i$;
8 **end**
9 **return** $R_{b'}$

The intermediate states of the registers are not randomized in Algorithm 13 and would require additional countermeasures to provide a secure implementation. For example, inexpensive solutions such as randomizing projective points [WMPW98] or Ebeid et al.'s blinding method for RSA [EL10] can be used (see Section 5.5.5). In the following, we show that, under certain assumptions, an implementation of Algorithm 13 is secure against first-order side-channel analysis. This means, that no intermediate variable in the implementation processes information about the secret key in clear, i.e. a random mask is applied in each intermediate state.

Lemma 1. If we assume that the values held in registers $\{R_0, R_1, R_2\}$ do not leak, an implementation of Algorithm 13 is resistant to first-order side-channel analysis.

Proof. It suffices to consider each intermediate state and verify that at least one random mask is applied. Verifying this for an entire group exponentiation would be tedious, but can be simplified if we consider two rounds of Algorithm 13. That is, if we consider round m, where $0 \leq m \leq n-2$, then the following operations are performed:

1. $R_2 \leftarrow R_{a_m} \cdot R_{\neg a_m}$	6. $R_2 \leftarrow R_{a_{m+1}} \cdot R_{\neg a_{m+1}}$
2. $\alpha \leftarrow b_m \oplus b'$	7. $\alpha \leftarrow b_{m+1} \oplus b_m$
3. $\beta \leftarrow \alpha \oplus a_m$	8. $\beta \leftarrow \alpha \oplus a_{m+1}$
4. $R_{a_m} \leftarrow {R_\beta}^2$	9. $R_{a_{m+1}} \leftarrow {R_\beta}^2$
5. $R_{\neg a_m} \leftarrow R_2$	10. $R_{\neg a_{m+1}} \leftarrow R_2$

Let proposition $\mathcal{P}(n)$ be that round $n > 0$ is resistant to first-order side-channel analysis for the n-th treated bit of the exponent. If we consider the first round, we wish to show $\mathcal{P}(1)$ is true and, in the above code fragment, b' is set to a random value from $\{0, 1\}$. Then, it is easy to see that:

- the results of the operations in lines 1, 4, 5, 6, 9 and 10 are dependent on the random values $\{R_0, R_1, R_2\}$.
- the results of the operations in lines 2, 3, 7 and 8 are uniformly distributed on $\{0, 1\}$.

If we assume that $\mathcal{P}(m)$ is true for $m \in \{1, \ldots, n\}$, then we consider $\mathcal{P}(n+1)$ where b' is set to b_n. As b_n is one share of a previously treated exponent bit, it is indistinguishable from a random value from $\{0, 1\}$. The above statements regarding the results of the operations apply. Hence, by induction we have shown $\mathcal{P}(n)$ is true for all $n > 0$. To complete the proof, we simply note that only half of the code fragment above will need to be considered in the last round. □

Remark 2. In [IIT03], the authors present the randomized addressing method (RA), in order to provide protection against ADPA and eliminate the correlation between the exponent bit and the register where an operation is stored. We do not aim to provide protection only against ADPA. Our goal is to perform operations on different

exponent shares, in a way that an adversary would need a combination of leakages (like Higher Order DPA combined with template attacks) in order to recover the exponent. For instance, in Algorithm 13, knowledge of a_i OR b' OR the register $R[a_i]$, would not give enough information to recover k_i. The adversary would need to know *all* those three values. Moreover, the way Algorithm 13 is structured, shows that it can tolerate zeroes as most significant bits of the scalar and therefore, it can hide the length of the scalar. The interested reader can run iterations of the algorithm for $a_{n-1} = b_{n-1} = 0$ or $a_{n-1} = b_{n-1} = 1$, meaning that $k_{n-1} = 0$, and verify that indeed the algorithm will perform all the operations and actually give the correct result.

5.3.2 Using Inverses

In this section we propose an algorithm more suited to groups where inversions can be readily computed. Le et al. [LTT15] propose a straightforward variant of the Montgomery powering ladder that requires the computation of inverses. They note that (5.2) can be rewritten as

$$(L_j, H_j) = \begin{cases} (H_j - 1, L_{j+1} + H_{j+1}) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1}, L_j + 1) & \text{if } k_j = 1\,. \end{cases} \tag{5.6}$$

This leads to the definition of Algorithm 14. If we let $T_{0,j} = L_j$ and $T_{1,j} = H_j$, or $T_{0,j} = H_j$ and $T_{1,j} = L_j$ and store the ordering in another variable we can rewrite (5.6) as

$$(T_{0,j}, T_{1,j}) = \begin{cases} (L_j, H_j) & \text{if } k_j = 0 \\ (H_j, L_j) & \text{if } k_j = 1 \end{cases} = \begin{cases} (L_{j+1} + H_{j+1}, L_j - 1) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1}, L_j + 1) & \text{if } k_j = 1\,. \end{cases} \tag{5.7}$$

This leads to Algorithm 15.

Algorithm 14: Variant with Inverses I

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = \sum_{i=0}^{n-1} k_i\, 2^i$
Output: x^κ

$R_0 \leftarrow 1_\mathbb{G}$; $R_1 \leftarrow x$;
$U_0 \leftarrow x^{-1}$; $U_1 \leftarrow x$;
for $i = n - 1$ **down to** 0 **do**
  $R_{\neg k_i} \leftarrow R_{k_i} \cdot R_{\neg k_i}$;
  $R_{k_i} \leftarrow R_{\neg k_i} \cdot U_{k_i}$;
end
return R_0

Algorithm 15: Variant with Inverses II

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = \sum_{i=0}^{n-1} k_i\, 2^i$
Output: x^κ

$R_0 \leftarrow 1_\mathbb{G}$; $R_1 \leftarrow x$;
$U_0 \leftarrow x^{-1}$; $U_1 \leftarrow x$;
for $i = n - 1$ **down to** 0 **do**
  $R_0 \leftarrow R_0 \cdot R_1$;
  $R_1 \leftarrow R_0 \cdot U_{k_i}$;
end
return $R_{\neg k_0}$

Following the previous notation, we notice that $T_{0,j}$ should contain the sum of the registers in the previous round.[2] Therefore, (5.7) can be rewritten as follows:

$$(T_{0,j}, T_{1,j}) = \begin{cases} (T_{b',j+1} + T_{\neg b',j+1}, T_{0,j} - 1) & \text{if } k_j = b' = 0\,, \\ (T_{b',j+1} + T_{\neg b',j+1}, T_{0,j} + 1) & \text{if } k_j = b' = 1\,. \end{cases} \tag{5.8}$$

We note that to treat k_{j+1}, $b' = k_j$. However, if we let $k_j = a_j \oplus b_j$, for $a_j, b_j \in \{0,1\}$ and $h = a_j \oplus b_j \oplus b_{j-1}$, we can modify (5.8) as follows:

$$(T_{0,j}, T_{1,j}) = \begin{cases} (T_{\neg h,j+1} + T_{h,j+1}, T_{0,j} - 1) & \text{if } a_j = b_j\,, \\ (T_{\neg h,j+1} + T_{h,j+1}, T_{0,j} + 1) & \text{if } a_j = \neg b_j\,. \end{cases} \tag{5.9}$$

By using the above equations as exponents of x, we can define Algorithm 16.

Algorithm 16: Montgomery Ladder with XOR-Split Exponent II

Input: $x \in \mathbb{G}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$, $r \in_R \mathbb{Z}$
Output: x^κ where $\kappa = A \oplus B$

$R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow 1_{\mathbb{G}}$; $U_0 \leftarrow x$; $U_1 \leftarrow x^{-1}$;
$b' \xleftarrow{R} \{0,1\}$; $R_{\neg b'} \leftarrow x$;
for $i = n-1$ **down to** 0 **do**
  $R_0 \leftarrow R_{b_i \oplus b'} \cdot R_{(b_i \oplus b') \oplus a_i}$;
  $R_1 \leftarrow R_0 \cdot U_{b_i}$;
  $b' \leftarrow b_i$;
end
return $R_{b'}$

Algorithm 16 follows the same sequence of instructions as the MPL. Its correctness can be verified by the fact that at every round the difference $R_0/R_1 = x$ or $R_1/R_0 = x$, as for the usual ladder step. The advantage of Algorithm 16 compared to Algorithm 13, and consequently previously proposed algorithms by Izumi et al. [IISO10], is the elimination of the auxiliary register R_2. Instead, the auxiliary registers U_0, U_1 manipulate the known fixed value x or x^{-1} for computational purposes, and they do not require additional computational power or updates when the algorithm is executed.

Lemma 2. If we assume that the values held in registers $\{R_0, R_1, R_2\}$ do not leak, an implementation of Algorithm 16 is resistant to first-order side-channel analysis.

Proof. It suffices to consider each intermediate state and verify that at least one random mask is applied. Verifying this for an entire group exponentiation would be

[2] The algorithms are left-to-right, so $j+1$ indicates the round preceding j.

tedious, but can be simplified if we consider two rounds of Algorithm 16. That is, if we consider round m, where $0 \leq m \leq n-2$, then the following operations are performed:

1. $\alpha \leftarrow b_m \oplus b'$	5. $\alpha \leftarrow b_{m+1} \oplus b_m$
2. $\beta \leftarrow \alpha \oplus a_m$	6. $\beta \leftarrow \alpha \oplus a_{m+1}$
3. $R_0 \leftarrow R_\alpha \cdot R_\beta$	7. $R_0 \leftarrow R_\alpha \cdot R_\beta$
4. $R_1 \leftarrow R_0 \cdot U_{b_m}$	8. $R_1 \leftarrow R_0 \cdot U_{b_{m+1}}$

Let the proposition $\mathcal{P}(n)$ be that round $n > 0$ is resistant to first-order side-channel analysis for the n-th treated bit of the exponent. If consider the first round, we wish to show $\mathcal{P}(1)$ is true and, in the above code fragment, b' is set to a random value from $\{0, 1\}$. Then, it is easy to see that:

- the results of the operations in lines 3, 4, 7 and 8 are dependent on the random values $\{R_0, R_1, R_2\}$.
- the results of the operations in lines 1, 2, 5 and 6 are uniformly distributed on $\{0, 1\}$.

If we assume that all $\mathcal{P}(m)$ is true for $m \in \{1, \ldots, n\}$, then we consider $\mathcal{P}(n+1)$ where b' is set to b_n. As b_n is one share of a previously treated exponent bit, it is indistinguishable from a random value from $\{0, 1\}$. The above statements regarding the results of the operations apply. Hence, by induction we have shown $\mathcal{P}(n)$ is true for all $n > 0$. To complete the proof, we simply note that only half of the code fragment above will need to be considered in the last round. □

Remark 3. All algorithms are presented with focus on the XOR-split exponent technique. In the actual implementation, we need to make sure that the order of computing the XOR-values by the compiler, will not cause another side-channel leakage. For instance, at Step 6 of Algorithm 13, the register $R_{(b_i \oplus b') \oplus a_i}$ is manipulated. A compiler could compute $b_i \oplus a_i$ first, as it is only obliged to provide something functionally equivalent. Therefore, precomputing $temp_i = b\prime \oplus b_i$ and then performing the multiplication of registers $R_{temp_i} \cdot R_{temp_i \oplus a_i}$ would prevent the implementation from leaking the value $b_i \oplus a_i$ in some way. This remark holds for all the algorithms presented in this chapter.

5.3.3 Joye's Add-Always Algorithm

Joye has proposed highly regular exponentiation algorithms [Joy07], where an exponentiation algorithm executes in a deterministic sequence of operations for an exponent of a given bit length. In this section we shall consider Joye's Add-Always algorithm as shown in Algorithm 17. Summarizing the reasoning given by Joye: Let

$\sum_{j=0}^{t-1} k_j\, 2^j$ with $k_j \in \{0,1\}$ be the binary expansion of κ. Then

$$x^\kappa = \prod_{j=0}^{t-1} x^{(k_j\, 2^j)} = \prod_{j=0}^{t-1} {\beta_j}^{k_j} \quad \text{where } \beta = x^{2^j}\,.$$

For $\ell \geq 0$, we define $U_\ell = \prod_{j=0}^{\ell} {\beta_j}^{k_j}$ and $T_\ell = \frac{\beta_{\ell+1}}{U_\ell}$. Then

$$U_\ell = \prod_{j=0}^{\ell} {\beta_i}^{k_i} = {\beta_\ell}^{k_\ell} U_{\ell-1} = {U_{\ell-1}}^{(1+k_\ell)} {T_{\ell-1}}^{k_\ell} \quad \text{and}$$
$$T_\ell = \frac{\beta_{\ell+1}}{U_\ell} = \frac{{\beta_\ell}^2}{{\beta_\ell}^{k_\ell} U_{\ell-1}} = {T_{\ell-1}}^{2-k_\ell} {U_{\ell-1}}^{1-k_\ell}\,.$$

This results in Algorithm 17:

$$U_\ell = \begin{cases} U_{\ell-1} & \text{if } k_\ell = 0\,, \\ {U_{\ell-1}}^2 T_{\ell-1} & \text{if } k_\ell = 1\,. \end{cases} \quad \text{and} \quad T_\ell = \begin{cases} U_{\ell-1} {T_{\ell-1}}^2 & \text{if } k_\ell = 0\,, \\ T_{\ell-1} & \text{if } k_\ell = 1\,. \end{cases} \tag{5.10}$$

Algorithm 17: Joye's Add-Always Algorithm

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = \sum_{i=0}^{n-1} k_i\, 2^i$
Output: x^κ

1 $R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow x$;
2 **for** $i = 0$ **to** $n-1$ **do**
3 $\quad R_{\neg k_i} \leftarrow {R_{\neg k_i}}^2$;
4 $\quad R_{\neg k_i} \leftarrow R_{\neg k_i} \cdot R_{k_i}$;
5 **end**
6 **return** R_0

Algorithm 18 can be defined with the input and intermediate states blinded by a random element $\alpha \in_R \mathbb{G}$. For any $\alpha \in_R \mathbb{G}$, Equation (5.10) can be written as:

$$\alpha U_\ell = \begin{cases} \alpha\, U_{\ell-1} \\ (\alpha\, U_{\ell-1})^2\, \alpha^{-1} T_{\ell-1} \end{cases} \quad \alpha^{-1} T_\ell = \begin{cases} \alpha\, U_{\ell-1}\, (\alpha^{-1} T_{\ell-1})^2 & \text{if } k_\ell = 0\,, \\ \alpha^{-1} T_{\ell-1} & \text{if } k_\ell = 1\,. \end{cases} \tag{5.11}$$

The same sequence of instructions as for the Montgomery powering ladder are followed, where the two registers are blinded by α. The advantage of Algorithm 18 compared to the previously proposed algorithms is that the intermediate states can be blinded with a random value at the cost of an inversion.

Algorithm 18: Blinded Add-Always Algorithm with XOR-Split Exponent

Input: $x \in \mathbb{G}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$
Output: x^κ where $\kappa = A \oplus B$

$\alpha \xleftarrow{R} \{\mathbb{G}\}$; $R_0 \leftarrow \alpha$; $R_1 \leftarrow \alpha^{-1}$; $R_2 \leftarrow 1_\mathbb{G}$;
$b' \xleftarrow{R} \{0,1\}$; $r' \leftarrow b'$; $R_{\neg b'} \leftarrow R_{\neg b'} \cdot x$;
for $i = 0$ **to** $n-1$ **do**
  $h \leftarrow (b_i \oplus b') \oplus a_i$;
  $R_2 \leftarrow {R_{\neg h}}^2 \cdot R_h$;
  $R_{a_i} \leftarrow R_h$;
  $R_{\neg a_i} \leftarrow R_2$;
  $b' \leftarrow b_i$
end
$R_{r' \oplus b'} \leftarrow \alpha^{-1} R_{r' \oplus b'}$;
$R_{\neg(r' \oplus b')} \leftarrow \alpha\, R_{\neg(r' \oplus b')}$;
return $R_{b'}$

5.3.4 Boolean Scalar Splitting Algorithms

In the above, we define group exponentiations applicable to any multiplicatively written group $\mathbb{G}$. However, specific groups may have particular characteristics, which means the algorithms above are not suitable as described. In this section, we discuss the algorithms in the context of a group formed from the points on an elliptic curve. Following the definitions from Chapter 2, we consider an elliptic curve $\mathcal{E}$ defined over a finite field $\mathbb{F}_q$, for a large prime q. The set $\mathcal{E}(\mathbb{F}_q)$ is defined as the set of points $\mathcal{E}(\mathbb{F}_q) = \{(x, y) \in \mathcal{E} \,|\, x, y \in \mathbb{F}_q\} \cup \{\boldsymbol{O}\}$, where $\mathcal{E}(\mathbb{F}_q)$ forms an Abelian group under the chord-and-tangent rule and $\boldsymbol{O}$ is the identity element. The scalar multiplication of a given point is a group exponentiation in $\mathcal{E}$ that uses elliptic curve arithmetic, i.e. addition between points or scalar multiplication $[\kappa]\,\boldsymbol{P}$ for some integer $\kappa < |\mathcal{E}|$, and, as we have seen earlier in this thesis, it is an important part of many cryptographic algorithms.

The algorithms presented above cannot be securely implemented as described because of the neutral element. The neutral element $1_\mathbb{G}$ is represented in $\mathcal{E}$ as the point at infinity $\boldsymbol{O}$; it cannot be manipulated in a regular way. That is, one would typically be obliged to test for a numerical representation of $\boldsymbol{O}$ and conduct a different operation if it is detected. In practice, one would implement the algorithm such that the most significant bit (assumed to be set to one) is already treated by the pre-processing. For example, Algorithm 13 can be implemented as shown in Algorithm 19, and Algorithm 16 as shown in Algorithm 20.

The change is not so simple for Joye's Add-Always algorithm, as any special

Algorithm 19: MPL with XOR-Split Scalar on an EC

Input: $\mathcal{E}, \mathbb{F}_q, \boldsymbol{P} \in \mathcal{E}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$;

$b' \xleftarrow{R} \{0,1\}$;
$\boldsymbol{R}_{\neg b'} \leftarrow 2\,\boldsymbol{P}$;

for $i = n-2$ **down to** 0 **do**
  $\boldsymbol{R}_2 \leftarrow \boldsymbol{R}_{a_i} + \boldsymbol{R}_{\neg a_i}$;
  $\boldsymbol{R}_{a_i} \leftarrow 2\,\boldsymbol{R}_{(b_i \oplus b') \oplus a_i}$;
  $\boldsymbol{R}_{\neg a_i} \leftarrow \boldsymbol{R}_2$;
  $b' \leftarrow b_i$;
end

return $\boldsymbol{R}_{b'}$

Algorithm 20: MPL with XOR-split scalar II on an EC

Input: $\mathcal{E}, \mathbb{F}_q, \boldsymbol{P} \in \mathcal{E}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
$\boldsymbol{U}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{U}_1 \leftarrow -\boldsymbol{P}$;

$b' \xleftarrow{R} \{0,1\}$;
$\boldsymbol{R}_{\neg b'} \leftarrow 2\,\boldsymbol{P}$;
for $i = n-2$ **down to** 0 **do**
  $\boldsymbol{R}_0 \leftarrow \boldsymbol{R}_{b_i \oplus b'} + \boldsymbol{R}_{(b_i \oplus b') \oplus a_i}$;
  $\boldsymbol{R}_1 \leftarrow \boldsymbol{R}_0 + \boldsymbol{U}_{b_i}$;
  $b' \leftarrow b_i$;
end

return $\boldsymbol{R}_{b'}$

treatment for $\boldsymbol{O}$ could reveal the number of least-significant bits (LSB) of the scalar set to zero. Most applications check if the point entered to the scalar multiplication is the point at infinity and perform a different operation if this is the case. It might also be that the additions are performed with projective coordinates, where the point at infinity will throw an exception. When the algorithm starts with the LSB, there is the change that the scalar bit is 0 (which is not the case for the MSB, since it is commonly set to 1). In those cases, the algorithm will return the same value $\boldsymbol{P}$, as long as a zero is encountered, revealing the number of least-significant zero scalar bits. To prevent this Joye proposes changing line 1 in Algorithm 17 to $R_0 \leftarrow \boldsymbol{P}$; $R_1 \leftarrow \boldsymbol{P}$ and inserting $R_{k_0} \leftarrow R_{k_0} - \boldsymbol{P}$ between line 4 and 5 [Joy07]. Alternatively, one could use a random point, as shown in Algorithm 21, where initializing the points $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$ to random points removes the need to treat the neutral element $\boldsymbol{O}$ differently. While $\boldsymbol{M}$ is defined as a random point, in practice this could be an arbitrarily chosen point that is randomized when it is used. That is, a fixed affine point that is used as randomized projective point [WMPW98].

Algorithm 21: Blinded Joye's Add-Always Algorithm with XOR-Split Joye on an EC

Input: $\boldsymbol{P} \in \mathcal{E}$, $\mathbf{M}$ a random point in $\mathcal{E}$, , n-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$

Output: $\boldsymbol{Q} = \kappa\, \boldsymbol{P}$ where $\kappa = A \oplus B$

1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{M}$; $\boldsymbol{R}_1 \leftarrow -\boldsymbol{M}$; $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$;

2 $b' \xleftarrow{R} \{0,1\}$; $r' \leftarrow b'$; $\boldsymbol{R}_{\neg b'} \leftarrow \boldsymbol{R}_{\neg b'} + \boldsymbol{P}$;

3 **for** $i = 0$ **to** $n-1$ **do**

4 $\quad h \leftarrow (b_i \oplus b') \oplus a_i$;

5 $\quad \boldsymbol{R}_2 \leftarrow 2\, \boldsymbol{R}_{\neg h} + \boldsymbol{R}_h$;

6 $\quad \boldsymbol{R}_{a_i} \leftarrow \boldsymbol{R}_h$;

7 $\quad \boldsymbol{R}_{\neg a_i} \leftarrow \boldsymbol{R}_2$;

8 $\quad b' \leftarrow b_i$

9 **end**

10 $\boldsymbol{R}_{r' \oplus b'} \leftarrow \boldsymbol{R}_{r' \oplus b'} - \boldsymbol{M}$;

11 $\boldsymbol{R}_{\neg(r' \oplus b')} \leftarrow \boldsymbol{R}_{\neg(r' \oplus b')} + \boldsymbol{M}$;

12 **return** $\boldsymbol{R}_{b'}$

The correctness of all the algorithms of this section used for scalar multiplication elliptic curves has been verified by Sage and the scripts are included in Appendix A 7.3.

5.4 Applying Exponent Splitting to ECDSA

In this section we describe how XOR exponent splitting can be transformed to a multiplicative split of the secret scalar in order to increase the security of an implementation of ECDSA. We consider this scalar multiplication for use with ECDSA separately as there are further security considerations in addition to those considered in Section 5.3.

For ECDSA, given the base point $\boldsymbol{P} = (x, y)$ for an EC $\mathcal{E}$ over $\mathbb{F}_q$, with private key d and hash function h, the signer who wants to sign a message m picks a random $\kappa < |\mathcal{E}| = n$ (where $n = |\mathcal{E}|$ is the order of the curve) and computes

$$r \overset{x}{\leftarrow} [\kappa]\boldsymbol{P} \text{ and } s \leftarrow \kappa^{-1}\left(h(m) + d\,r\right) \bmod |\mathcal{E}|\,.$$

We denote the extraction of the x-coordinate of a point and its assignment to a variable by $\overset{x}{\leftarrow}$. The signature of m is the pair: $\{r, s\}$. We note that the security of this signature scheme relies on the random value κ remaining unknown to an attacker. Moreover, the nonce κ should be used only once and randomly generated for every new signature, otherwise the private key d can be trivially derived [Nat09].

The verifier, after receiving the signature pair $\{r, s\}$ needs to verify that this is valid. First, he verifies that r, s belong to the interval $[1, n-1]$. Then he computes $e = hash(m)$ and $w = s^{-1} \pmod n$, $u_1 = ew$ and $u_2 = rw \pmod n$. Knowing the public key Q_s of the signer, he can calculate $V = u_1 \cdot G + u_2 \cdot Q_s$. If $V = \mathcal{O}$, then the signature is rejected. Converting the x-coordinate of V to an integer $\bar{x}$, the verifier can compute if $u = \bar{x} \pmod n = r$. If the last equation holds, he can accept the signature.

5.4.1 Securing the Scalar Multiplication

The scalar multiplication $[\kappa]\boldsymbol{P}$ during the computation of r is a critical operation in ECDSA from a side-channel point of view, because an attacker only needs to derive a small number of bits of κ to obtain the secret key d. For example, a lattice-based analysis can reveal the private key from the knowledge of some bits of the scalar κ [HGS01]. In practice, one could use the knowledge of one bit from around 30 signatures or eleven bits from one signature [NNTW05]. Similarly, one needs to prevent an adversary from determining the bit length of the scalar used to prevent a lattice attack.

When implementing a scalar multiplication, one could use any of the algorithms described in the previous section, with extra care required to prevent the bit length of the scalar from leaking. Algorithm 21 requires no further modification, as if the most-significant bit is zero the algorithm still functions correctly. However, Algorithms 19 and 20 require that the most-significant bit of the scalar is set to one and need further modification to ensure that the bit length of the scalar is not revealed.

In implementing Algorithms 19 or 20, one could add a multiple of the order of the group to the scalar to hide its length [Cor99]. While this would seem to make the algorithms redundant, one could choose a small multiple of the order of the group without considering the longest run of ones or zeros in the bitwise representation of the order. Indeed, one could multiply the order of the group by a power of two such that increasing the bit length is impossible or fixed to a one bit increase. That is, ensure that the carries produced by adding the multiple of the order of the group do not extend the bit length or always extend the bit length. Given the long runs of ones or zeros typically seen in the bitwise representation of the order of many groups formed from the points of an EC, the increase in bit length should be very small [Nat09, BCLN16, Ham15, Ber06]. For example, if we consider P192 defined by NIST [NIS13], where the elliptic curve is defined over $\mathbb{F}_p$, with $p = 2^{192} - 2^{64} - 1$, then the 128 most-significant bits of p are set to one. If twice the order is added to a nonce then a carry from the most-significant bits will be produced with an overwhelming probability, thus always producing a nonce of 194 bits. However, in some cases, one may wish to ensure that the carry has been produced to prevent a theoretical reduction in the security of a system.

Alternatively, one could add some more logical functions to allow for the most significant bit to be set to zero. In Algorithms 22 and 23 we show how this could be done for Algorithms 19 and 20, respectively. In both cases we set the variable m to the value three, while the most-significant bits of the scalar, $A \oplus B$, are set to zero, allowing the output of operations to be diverted into $\boldsymbol{R}_2$. In Algorithm 23, we do this for both operations in the loop and keep the initial state of $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$ until after the first bit set to one is treated. Then Algorithm 23 proceeds in the same way as Algorithm 20. Algorithm 22 starts in the same way, using m to divert the output of the operations into $\boldsymbol{R}_2$. When the most-significant bit of the scalar, $A \oplus B$, set to one is treated the output stored in $\boldsymbol{R}_2$ is copied to either $\boldsymbol{R}_0$ or $\boldsymbol{R}_1$. Then Algorithm 22

proceeds in the same way as Algorithm 19.

Algorithm 22: Montgomery Ladder with Fixed-Length XOR-split scalar on an EC

Input: $\mathcal{E}, \mathbb{F}_q, \boldsymbol{P} \in \mathcal{E}$, ℓ-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$ where $n \geq \ell$

Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$;

$b' \xleftarrow{R} \{0,1\}$; $h \leftarrow 1$; $m \leftarrow 3$;

for $i = n-1$ **down to** 0 **do**

  $\boldsymbol{R}_2 \leftarrow \boldsymbol{R}_{a_i} + \boldsymbol{R}_{\neg a_i}$;

  $\boldsymbol{R}_{((1+a_i)\vee m)-1} \leftarrow 2\,\boldsymbol{R}_{(h\oplus b_i \oplus b')\oplus a_i}$;

  $h = m \wedge 1$;

  $m = 3\,((h \wedge \neg a_i) \oplus (h \wedge b_i))$;

  $\boldsymbol{R}_{((1+\neg a_i)\vee m)-1} \leftarrow \boldsymbol{R}_2$;

  $b' \leftarrow b_i$;

end

return $\boldsymbol{R}_{b'}$

Algorithm 23: Montgomery Ladder with Fixed-Length XOR-split scalar II on an EC

Input: $\mathcal{E}, \mathbb{F}_q, \boldsymbol{P} \in \mathcal{E}$, ℓ-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$ where $n \geq \ell$

Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;

$\boldsymbol{U}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{U}_1 \leftarrow -\boldsymbol{P}$;

$b' \xleftarrow{R} \{0,1\}$; $n \leftarrow n$; $h \leftarrow 1$; $m \leftarrow 3$;

$\boldsymbol{R}_{\neg b'} \leftarrow 2\,\boldsymbol{P}$;

for $i = n-1$ **down to** 0 **do**

  $\boldsymbol{R}_{(1\vee m)-1} \leftarrow \boldsymbol{R}_{b_i \oplus b'} + \boldsymbol{R}_{(b_i \oplus b')\oplus a_i}$;

  $\boldsymbol{R}_{(2\vee m)-1} \leftarrow \boldsymbol{R}_0 + \boldsymbol{U}_{b_i}$;

  $b' \leftarrow (b_i \wedge \neg n) \oplus (b' \wedge n)$;

  $s = ((m \wedge \neg a_i) \oplus (m \wedge b_i))$;

  $m = 3\,s$;

end

return $\boldsymbol{R}_{b'}$

5.4.2 Using a Nonce as Two Shares

In this section we describe how one could generate an ECDSA signature where the nonce is only present as two shares. That is, one could generate two random values to represent the two shares of the nonce, and the actual value of the nonce would never appear in the signing device.

We assume that the first half of the signature r has been generated using one of the scalar multiplication algorithms given above, using shares A and B where the nonce $\kappa = A \oplus B$. To compute the inverse of κ modulo $|\mathcal{E}|$ we need to change the way that κ is shared between two variables without revealing any information on κ.

Goubin defined a secure means of modifying two shares such that the secret value is no longer given by the XOR of the shares, but by subtracting one from the other [Gou01]. That is, the relationship $\kappa = A \oplus B$ becomes $\kappa \equiv A' - B' \pmod{|\mathcal{E}|}$. The essential observation of Goubin was that the function

$$\Phi(a, b) : \mathbb{Z}^2 \longrightarrow \mathbb{Z} : a, b \longmapsto (a \oplus b) + b \tag{5.12}$$

is affine over $\mathbb{F}_2$. If we consider α, β as Boolean shares of x, where $x = \alpha \oplus \beta$, then, as defined by Goubin,

$$x + \alpha = \beta \oplus \Phi(\beta, \gamma) \oplus \Phi(\beta, \gamma \oplus \alpha) \ , \tag{5.13}$$

for a random variable γ. This is not side-channel resistant in $\mathbb{Z}$ because of the carries produced by the addition operations. One needs to use a group $\mathbb{Z}_{2^k}$, for some $k \in \mathbb{Z}_+$, and have γ be a random value from $\mathbb{Z}_{2^k}$. However, we wish to change a Boolean share into an additive share in the group $\mathbb{Z}_{|\mathcal{E}|}$ where (5.13) is not valid.

We can modify the shares such that Goubin's mask conversion can be used adding a random multiple of the order of the group as proposed by Coron for other algorithms [Cor99]. One can generate a random integer $\ell \in \mathbb{Z}_{2^n}$ for some convenient bit length n. The Boolean shares $(A = \kappa \oplus B, B)$ can be modified to (A', B') where

$$(A', B') = (A \oplus (\ell|\mathcal{E}| + B) \oplus B, \ell|\mathcal{E}| + B) \ . \tag{5.14}$$

We note that this gives $A' = \kappa \oplus (\ell|\mathcal{E}| + B)$. If we consider the bit length of $|\mathcal{E}|$ is m then the above modified shares can be used with (5.13) using the group $\mathbb{Z}_{2^{n+m}}$ to change the shares (A', B') to (A'', B'') where

$$(A'', B'') = (\kappa + \ell|\mathcal{E}| + B, \ell|\mathcal{E}| + B) \ . \tag{5.15}$$

Then A'' and B'' can be reduced modulo $|\mathcal{E}|$ to provide additively split shares of κ in $\mathbb{Z}_{|\mathcal{E}|}$. We can generate a random integer $\omega < |\mathcal{E}|$ and change the shares to a multiplicative split (A''', B''') where

$$(A''', B''') = (\omega\, A'' - \omega\, B'' \bmod |\mathcal{E}|, \omega) \ . \tag{5.16}$$

We note that this gives $A''' \equiv \omega\kappa \pmod{|\mathcal{E}|}$. The second part of the signature can be generated from r by computing

$$s \leftarrow B''' \left(A'''^{-1} \left(h(m) + d\, r \right) \right) \bmod |\mathcal{E}| \,.$$

Thus, using the above, one can generate an ECDSA signature where the nonce κ only exists as two shares. An explicit algorithm requiring 16 operations is given in Algorithm 24.

Algorithm 24: Secure Boolean-to-Multiplicative Masking

Input: $A, B \in \mathbb{Z}_{2^n}$, where $\kappa = A \oplus B$, $\kappa \in Z_{|\mathcal{E}|}$ and n is the bit length of $|\mathcal{E}|$, random value $\gamma \in \{0, \ldots, 2^n - 1\}$, random value $\ell \in \{0, \ldots, 2^m - 1\}$ where m is the bit length of one computer word and random value $\omega \in \{1, \ldots, |\mathcal{E}|\}$.
Output: $\kappa\,\omega \bmod |\mathcal{E}|$.

1 $x_1 \leftarrow \ell \times |\mathcal{E}|$;
2 $B' \leftarrow x_1 + B$;
3 $x_2 \leftarrow B' \oplus B$;
4 $A' \leftarrow A \oplus x_2$;
5 $x_3 \leftarrow A' \oplus \gamma$;
6 $x_4 \leftarrow x_3 + \gamma$;
7 $x_5 \leftarrow B' \oplus \gamma$;
8 $x_6 \leftarrow A' \oplus x_5$;
9 $x_7 \leftarrow x_6 + x_5$;
10 $x_8 \leftarrow x_4 \oplus x_7$;
11 $x_9 \leftarrow A' \oplus x_8$;
12 $B'' \leftarrow B' \bmod |\mathcal{E}|$;
13 $A'' \leftarrow A' \bmod |\mathcal{E}|$;
14 $x_{10} \leftarrow B'' \times \omega \bmod |\mathcal{E}|$;
15 $x_{11} \leftarrow A'' \times \omega \bmod |\mathcal{E}|$;
16 $x_{12} \leftarrow x_{11} - x_{10} \bmod |\mathcal{E}|$;
return x_{12} ;

5.5 Security Evaluation

In this section, we provide a security evaluation of Algorithm 16 presented in this chapter in terms of security and their resistance to multiple forms of side-channel attacks. We note that similar analysis can be carried out for all exponent splitting variants presented in this work. Section 5.5.3 discusses the probing security of algorithms, using formal methods. Section 5.5.4 analyzes security in the noisy leakage model, using an information-theoretic approach.

5.5.1 Theoretical Comparison with the State-of-the-Art Algorithms

In this section, we compare our proposed algorithms with a selection of algorithms discussed in the previous sections.

The first block of algorithms in the table, contain exponentiation algorithms using the Montgomery power ladder without splitting the exponent (Algorithm 12), with additive splitting or with variations of XOR-splitting (Algorithms 13, 14, 15). Multiplicative or Euclidean splitting are not included in this table, because in terms of security they have the same side-channel resistance as the algorithm with additive splitting. In terms of performance, the number of operations is the same, unless the values $s^{k\prime}$ are precomputed and stored in memory.

The second block of algorithms refer to Joye's Add-Always algorithm (Algorithm 17). The blinded version of this algorithm (Algorithm 18) with XOR-split

Table (5.1) Comparison Table for scalar multiplication algorithms, where M indicates a multiplication, S a squaring, A a point addition, D point doubling, and I a modular inversion

Algorithm for n-bit scalar	♯ operations	registers	Hide length	ADPA	Inter. Values
Algorithm 12	$nM + nS$	2	✗	✗	✗
Additive Split [CJ01]	$2(nM + nS)$	2	✗	✗	✗
Algorithm 13	$nM + nS$	3	✓	✓	✗
Algor. 14, 15	$2nM$	4	✓	✓	✗
Algorithm 16	$2n \cdot M$	4	✓	✓	✗
Algorithm 17	$nM + nS$	2	✗	✗	✗
Algorithm 18	$(n+1)\ M + nS + 1I$	4	✓	✓	✓
Algorithm 8 [IIT03]	$(n-1)D+$ $(n-1)A + 1I$	3	✗	✓	✗
Algorithm 2 [IISO10]	$(n-1)D + (n-1)A$	3	✗	✓	✗
Algorithm 19	$(n-1)D + (n-1)A$	3	✗	✓	✗
Algorithm 20	$2(n-1)A$	4	✗	✓	✗
Algorithm 21	$(n+1)A + (n-1)D$	4	✓	✓	✓
Algorithm 22	$(n-1)A + (n-1)D$	5	✓	✓	✗
Algorithm 23	$(n-1)A + (n-1)D$	6	✓	✓	✗

exponent provides protection against ADPA, leakage of the length and intermediate values. Finally, the last block of algorithms summarizes the behavior of the corresponding scalar multiplication algorithms.[3]

5.5.2 Security Against Template Attacks

Template Attacks in general are a powerful attack technique, because they take advantage of most of the information available in the traces that are measured [CRR02]. As we saw earlier in this thesis, the attacker is assumed to possess a device similar to the device under attack, and he can build templates for some instructions knowing their power consumption characteristics on this device.

In a template-based DPA attack, the attacker matches the templates based on different key hypotheses with the recorded power traces [OM07]. The templates

[3]We do not count XORs, which can be implemented almost for “free" compared to the cost of multiplications (M), squarings (S) and modular inversions (I) in the chosen field or point additions (A) and doublings (D) on an elliptic curve. The subtraction of points on an elliptic curve has the same cost as an addition, so we do not count them separately.

that match best indicate the key. This type of attack is the best attack in an information theoretic sense and in a masking scheme, like ours, it would work if we build templates for each share a_i and b_i. Simultaneous knowledge of the two shares is necessary, because for every possible value of the key bit, there are two different possibilities for a_i, b_i that could give the same result, as follows:

$$k_i = 1 \Rightarrow (a_i = 0 \quad \text{and} \quad b_i = 1) \quad \text{or} \quad (a_i = 1 \quad \text{and} \quad b_i = 0)$$

$$k_i = 0 \Rightarrow a_i = b_i = 0 \quad \text{or} \quad a_i = b_i = 1$$

In operations where $b\prime$ is also involved, such as line 5 of Algorithm 13, line 4 of Algorithm 16 etc., we need to store the b_{i-1} value from the templates of the previous round. This procedure seems quite complex and unlikely to succeed in devices with small signal-to-noise ratio.

In Chapter 4 we presented Online Template Attacks (OTA) [BCP+17]. OTA use templates of the multiples of points on the curve $k\boldsymbol{P}$, for $k \in \mathbb{Z}$ and compares them with the result of a specific operation, for instance doubling, for every round of the algorithm. As explained previously, every unprotected implementation of scalar multiplication is vulnerable to OTA, unless the coordinates of the base point are randomized.[4] The algorithms presented in this chapter are secure against OTA, because the doubling (or not) of a specific point, does not reveal the bit of the scalar k_i. We demonstrate this claim, using two algorithms for case studies, one for the case of exponentiation and one for elliptic curves. Extension to other algorithms presented in the previous sections is straightforward.

Algorithm 13: In line 4 $R_{a_i} \leftarrow \left(R_{(b_i \oplus b') \oplus a_i}\right)^2$ the squaring is performed and we can make templates for the potential values R_0^2 and R_1^2, in order to find out which operation is most probable to take place. Let us assume that there is a pattern match of 99% with the value R_0^2. Even if we know with high probability which value is calculated at this step, we do not know which register will be used in the next step using OTA. Moreover, the index 0 can be derived from several combinations of a_i, b_i, b'. Namely, if $a_i = b_i = 0, b' = 0$, or $a_i = b_i = 1, b' = 0$ or $a_i = 0, b_i = 1, b' = 1$ or $a_i = 1, b_i = 0, b' = 1$. If the previous shares are known, therefore b' is known, there are still two possible values for a_i and b_i that can give the same result with equal probability. The success probability of OTA is then $1/2$ which is not more than a random guess.

Algorithm 19: Let us assume that templates on doubling showed that in line 6 of Algorithm 8, we have $2R_1$. The possible values of a_i, b_i, b' that could result in 1 are: $(a_i = b_i = 0, b' = 1)$, or $(a_i = b_i = 1, b' = 1)$ or $(a_i = 0, b_i = 1, b' = 0)$ or $(a_i = 1, b_i = 0, b' = 0)$. The success probability of OTA is again $1/2$ if the previous shares are known. We note here, that in both cases, knowledge of the previous bit is

[4] Although OTA is initially presented for the case of elliptic curves, the attack can be used against exponentiation algorithms as well, as long as templates for certain exponent values can be created.

not enough to give a success probability of $1/2$, but knowledge of at least one of the shares is needed.

This analysis shows that it would be impractical to create templates for all the possible values of the shares for a scalar of 192-bits or an exponent that is thousands bits long.

5.5.3 Provable Security Against First-Order Attacks

It is common for masking schemes to provide security proofs showing that every masked intermediate value is statistically independent of the secret (unmasked) value. When a secret value is masked by two shares, the scheme is considered secure if it can be proven to resist first-order attacks. More generally, a nth-order masking scheme can protect an implementation up to nth-order attacks.

For the algorithms presented in the previous sections using the countermeasure of Boolean splitting, the goal is to protect the exponent (or scalar) key bits $(k_{n-1}, k_{n-2}, ..., k_0)$ by using the XOR-mask $k_i = a_i \oplus b_i \forall i \in \{0, ..., n-1\}$. The goal of the adversary is, therefore, to recover the current bit k_i by observing the available intermediate values each time. During the design phase of all the algorithms, we prevented combined access to shares a_i and b_i. To detect potential 1st-order security flaws, we use the Lisp-based formal verification tool suggested by Coron [Cor17]. The tool can generate all possible tuples of intermediate values (with dimension less or equal to our order) that stem from the schemes and verifies the security properties using circuit transformations. The tool checks all possible tuples of intermediate values computed during a single iteration of the exponentiation algorithm. In order to integrate location leakage in the formal verification we do not focus solely on manipulated values but also on register indexes used during the every algorithm. Using the tool we have shown that Algorithms 13,16,18,19,20,21 have the 1-NI security property. Rephrasing, we show that any set of at most one intermediate variable can be perfectly simulated with at most one share of each input [BBD$^+$16].

5.5.4 MI-based Evaluation of Boolean Exponent Splitting

Having established that the proposed exponent splitting algorithms are 1-NI, we proceed to analyze the noise amplification stage of the proposed countermeasure. Analytically, we perform an evaluation of Boolean exponent splitting using the information-theoretic framework of Standaert et al. [SMY09]. We present the evaluation of our countermeasure as described by Algorithm 16. However, analogous approaches can be carried for all exponent splitting algorithms, yielding very similar results. Our analysis considers two sources of leakage, namely data-based leakage and location-based leakage (also known as address leakage). Using these two leakage sources, we demonstrate three possible attack paths against Algorithm 16,

covering all possible combinations between sources. Thus we show the noise amplification stage when only data-based leakage is exploited (data attack), when only location-based leakage is exploited (location attack) and finally the noise amplification stage when the adversary combines data and location leakage (hybrid attack).

Notation & MI Metric. In this section, random variables are denoted with capital letters. Instances of random variables and constant values are denoted with lowercase letters. Capital bold letters are used for random variable vectors and matrices and calligraphic font denotes sets. All simulations in this section are carried out with the identity leakage function. Observable data-based leakages of a certain intermediate value v are denoted using subscript L_v. Likewise, observable location-based leakages caused by accessing register R_i (where i the index) are denoted using subscript $L_{R\text{-}i}$. To distinguish between data-based leakage and location-based leakage we use superscript L^{data} and L^{loc}. In addition, we assume that different sources of leakage (data, location) have different noise levels i.e. we assume homoscedastic data noise $N^{data} \sim \mathcal{N}(0, \sigma^2_{data})$ and homoscedastic location noise $N^{loc} \sim \mathcal{N}(0, \sigma^2_{loc})$. We use the following formula to compute the MI metric.

$$MI(S;\mathbf{L}) = H[S] + \sum_{s\in\mathcal{S}} \Pr[s] \cdot \sum_{\mathbf{m}\in\mathcal{M}^d} \Pr[\mathbf{m}] \cdot \int_{\mathbf{l}\in\mathcal{L}^{(d+1)}} \Pr[\mathbf{l}|s,\mathbf{m}] \cdot \log_2 \Pr[s|\mathbf{l}] \;\; d\mathbf{l}$$

$$\text{where } \Pr[s|\mathbf{l}] = \frac{\sum_{\mathbf{m}^*\in\mathcal{R}} \Pr[\mathbf{l}|s,\mathbf{m}^*]}{\sum_{s^*\in\mathcal{S}} \sum_{\mathbf{m}^*\in\mathcal{R}} \Pr[\mathbf{l}|s^*,\mathbf{m}^*]} .$$

The random variable S denotes the secret exponent bit, $\mathbf{L}$ denotes the leakage vector and $\mathbf{M}$ is a d-dimensional randomness vector that we need to sum over.

Data dependent leakage attack. The first obvious way to recover k_{n-1} is by observing the data leakage of the values b_{n-1} and a_{n-1} at the same time. We run the algorithm for the first two rounds and note the intermediate values that can leak information.

Algorithm 25: Two iterations of Algorithm 16

1 $b' \in_R \{0,1\}, R_0$	
2 $i = n-1$	8 $i = n-2$
3 $b_m = b_{n-1} \oplus b'$;	9 $b_m = b_{n-2} \oplus b'$;
4 $a_m = b_m \oplus a_{n-1}$;	10 $a_m = b_m \oplus a_{n-2}$;
5 $R_0 = R_{b_m} \cdot R_{a_m}$;	11 $R_0 = R_{b_m} \cdot R_{a_m}$;
6 $R_1 = R_0 \cdot U_{b_{n-1}}$;	12 $R_1 = R_0 \cdot U_{b_{n-2}}$;
7 $b' = b_{n-1}$;	13 $b' = b_{n-2}$;

As can be observed in Algorithm 25, the value b_{n-1} is accessed in the first iteration ($i = n-1$) three times, once when b_m is calculated (line 1), once implicitly for the index of $U_{b_{n-1}}$ (line 4) and finally for b' (line 5). The value a_{n-1} is accessed

once during the first iteration ($i = n - 1$) and it is not used in the second iteration ($i = n - 2$). We notice that the value b_{n-1} is used implicitly again in the second iteration, since it is equal to b'. An attacker observing the power leakage of this algorithm should be able to probe at two different points in time, in order to observe both leakages $L^{data}_{a_{n-1}}$, $L^{data}_{b_{n-1}}$ and eventually the key, i.e. we conclude that a second-order attack is possible for this scheme. Note also that an adversary with horizontal exploitation capabilities can observe the leakage of b_{n-1} multiple times, average them by computing $\bar{L}^{data}_{b_{n-1}} = \frac{1}{4} * \sum_{j=1}^{4} L^{data}_{b_{n-1}}$ in order to reduce the noise level and finally perform a 2nd-order attack. The results of the MI evaluation are visible in Figure 5.1. As expected, the exponent splitting scheme performs noise amplification and has a different slope compared to an unprotected exponentiation (Algorithm 12). In addition, we observe the curve's horizontal shift to the right caused by the horizontal exploitation of the available leakage, i.e. we can quantify the effect of multiple leaky points for b_{n-1}.

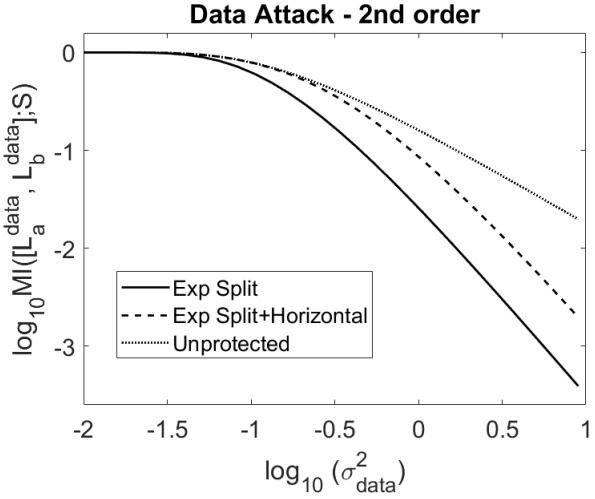

Figure (5.1) MI evaluation for Algorithm 16 exponent splitting, using a data leakage attack, with and without horizontal exploitation. Observed leakage vector $\mathbf{L} = [L^{data}_{a_{n-1}}, L^{data}_{b_{n-1}}]$

.

Location dependent leakage attack. Let us assume that the adversary can distinguish between the manipulation of registers according to which address is accessed, similar to the address-bit DPA attack described in [IISO10]. If he can distinguish between access of e.g. U_0 and U_1, a direct consequence is recovery of value b_{n-1}. To mount a successful attack against Algorithm 5 using solely location-based leakage, we need the simultaneous observation of the address of U_{i_1} and R_{i_2} and R_{i_3}, for

indexes $i_1 = b_{n-1}$ (line 4) and $i_2 = b_m$ (line 3) and $i_3 = a_m$ (line 3). Thus, in order to recover k_{n-1}, we need to observe the leakage vector $L^{loc} = [L^{loc}_{U\text{-}i_1}, L^{loc}_{R\text{-}i_2}, L^{loc}_{R\text{-}i_3}]$, i.e. perform a 3rd-order attack. The results are visible in Figure 5.2 where we can observe the noise amplification effect that increases the curve's slope. Naturally a 3rd-order attack using only location-based leakage tends to be less effective compared to the 2nd-order attack using only data-based leakage. However, depending on the device, exploiting the address dependency may be more effective than exploiting the data dependency, thus the 3rd-order attack can become more efficient if $\sigma_{data} > \sigma_{loc}$.

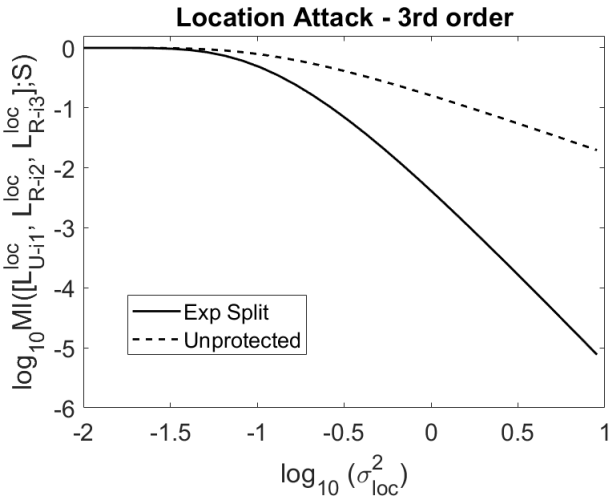

Figure (5.2) MI evaluation for Algorithm 16 exponent splitting, using a location leakage attack. Observed leakage vector $\mathbf{L} = [L^{loc}_{U\text{-}i_1}, L^{loc}_{R\text{-}i_2}, L^{loc}_{R\text{-}i_3}]$

.

Hybrid leakage attack. Last, we analyze the scenario at which the adversary can observe both data-based and location-based leakage. Using this information the adversary can use leakage vector $\mathbf{L} = [L^{data}_{a_{n-1}}, L^{loc}_{U\text{-}b_{n-1}}]$ to carry out a 2nd-order attack that uses data leakage to recover bit a_{n-1} and location leakage w.r.t. register U to recover bit b_{n-1}. Since data and location leakage imply different noise levels ($\sigma_{data} \neq \sigma_{loc}$ then we need to represent the available information using the 3D plot of Figure 5.3. The wave-like plot quantifies the attainable information w.r.t. a particular data and location noise level. Thus, it assists the side-channel evaluator to analyze the scheme's security in a more holistic way that factors in location leakage and demonstrates the trade-off between data noise and location noise.

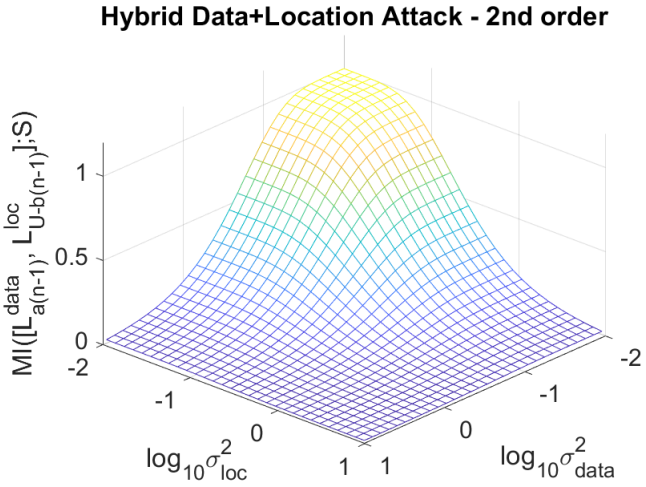

Figure (5.3) MI evaluation for Algorithm 16 exponent splitting, using a hybrid leakage attack. Observed leakage vector$\mathbf{L} = [L^{data}_{a_{n-1}}, L^{loc}_{U\text{-}b_{n-1}}]$

.

5.5.5 TVLA Results

In this section we will describe the results of applying Test Vector Leakage Assessment (TVLA) [GJJR11] to implementations of some of the algorithms above. We further describe modifications required to achieve a secure implementation where the hardware architecture can mean that variables that should leak independently at the same time, potentially unmasking a secret value [BGG+14].

Our implementations were developed using Xilinx's Zynq zc702 evaluation board [Xil]. The Zynq zc702 microprocessor contains two ARM7 cores and an FPGA fabric. We used one ARM7 core for our implementations, we clocked at 667 MHz, and the FPGA provided a means of triggering an oscilloscope at a convenient point in our implementations. We acquired a trace of the electromagnetic emanations around one of the coupling capacitors.

The test that we used from TVLA is to determine whether there are statistically significant differences in the mean traces of two sets of traces, one acquired with a fixed scalar and the other with random scalar. One would typically randomly interleave acquisitions so that environmental effects are the same for both sets and there are no erroneous indications of leakage, caused, for example, by the least significant bit of a variable used to count the number of acquisitions. In applying this, one would take two sets of data, and conduct Welch's t-test point-by-point to determine whether there is evidence against the null hypothesis that the sets are the same. We determine that leakage is present if we observe values above $\pm 6\,\sigma$ which gives the probability of indicating leakage where no leakage is present, often referred to as a Type I error, of approximately 1×10^5. The interested reader is referred to Goodwill

et al. [GJJR11] for a thorough description and Sections 3.6.1,6.3.1 of this thesis for specific details on applying TVLA to public key algorithms.

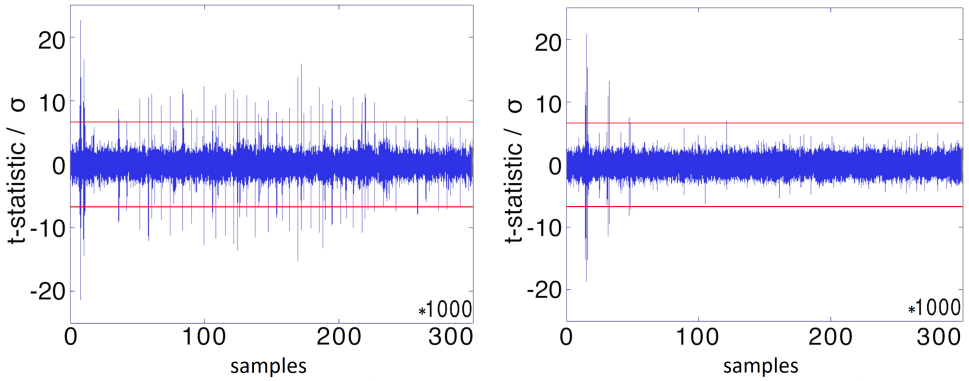

Figure (5.4) TVLA plots showing the unprotected implementation (left), the masking conducted before the execution of the scalar multiplication (right).

We made a straightforward implementation of Algorithm 19 using NIST's P192 curve and conducted a test where we compared a set of traces with a fixed scalar compared to a set of traces with a random scalar. The elliptic curve points were implemented as homogeneous projective points. We use the x and z-coordinates in conjunction with so-called x-only algorithms for point arithmetic [BJ02], as one would for an implementation of ECDH. The instantaneous electromagnetic emanations around the targeted capacitor were measured during the execution of the first 20 rounds of the implementation. The left trace in Figure 5.4 shows the result of a TVLA analysis with 1×10^3 traces where leakage can be seen in numerous places. A straightforward implementation of Algorithm 19 was tested in the same way. The algorithm is similar to that proposed by Izumi et al. [IIT03] but with masking conducted before the execution of the scalar multiplication, rather than on-the-fly. The resulting TVLA traces is shown in the right of Figure 5.4, where we note that significant leakage is present with 1×10^6 traces. This is caused by the microprocessor combining values held in registers because of the architecture chosen by the designers of the evaluation board [Xil].

A more secure implementation can be made by computing some of the required indices before the execution of the main loop of the scalar multiplication, as shown in Algorithm 26. We set C to $B \oplus \left\lfloor \frac{B}{2} \right\rfloor$ such that individual bits of B are masked by adjacent bits. The resulting TVLA trace is shown in the left Figure 5.5, where we observe that there is only one place where we see significant leakage with 1×10^6 traces. This leakage occurs because the initial state of $\{\boldsymbol{R}_0, \boldsymbol{R}_1\}$ contains $\{\boldsymbol{P}, 2\,\boldsymbol{P}\}$ in some random order. In the first loop of the scalar multiplication $\{\boldsymbol{R}_0, \boldsymbol{R}_1\}$ is overwritten with $\{2\,\boldsymbol{P}, 3\,\boldsymbol{P}\}$ or $\{3\,\boldsymbol{P}, 4\,\boldsymbol{P}\}$, in some random order, depending on whether the second most-significant bit of κ is set to 0 or 1, respectively. When $2\,\boldsymbol{P}$

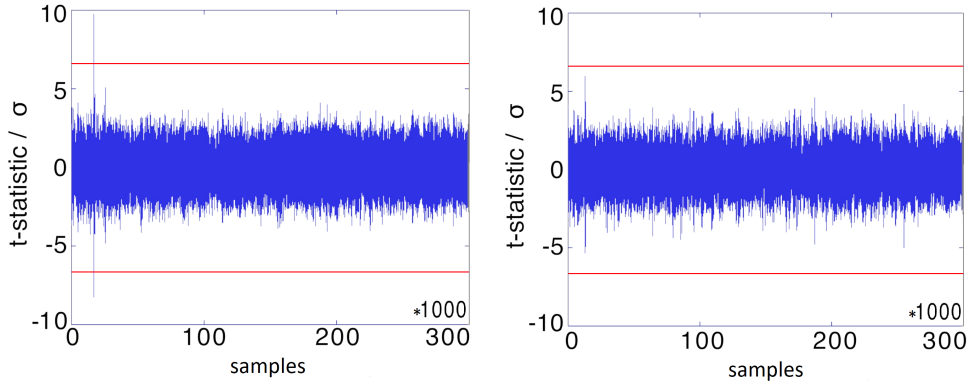

Figure (5.5) TVLA plots showing precomputation of some required indices before the execution of the main loop (left) and the fully secure implementation (right).

overwrites $2\,\boldsymbol{P}$ the side-channel leakage will be significantly different to any other possible combination, since the Hamming distance will be zero.

Algorithm 26: Montgomery Ladder with XOR-split scalar on an EC

Input: $\mathcal{E}$, $\mathbb{F}_q$, $\boldsymbol{P} \in \mathcal{E}$, $n-$bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$
Uses: $C = \sum_{i=0}^{n-1} c_i\, 2^i$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$;
$C \leftarrow B \oplus \left\lfloor \frac{B}{2} \right\rfloor$;
$b' \leftarrow b_{n-1}$;
$\boldsymbol{R}_{\neg b'} \leftarrow 2\,\boldsymbol{P}$;
for $i = n-2$ **down to** 0 **do**
  $\boldsymbol{R}_2 \leftarrow \boldsymbol{R}_{a_i} + \boldsymbol{R}_{\neg a_i}$;
  $\boldsymbol{R}_{a_i} \leftarrow 2\,\boldsymbol{R}_{a_i \oplus c_i}$;
  $\boldsymbol{R}_{\neg a_i} \leftarrow \boldsymbol{R}_2$;
end
return $\boldsymbol{R}_{b_0}$

A fully secure implementation can be achieved by randomizing the point produced by the doubling operation, by multiplying the x and z-coordinate of the resulting point by a random value. In implementing Algorithm 26, this was achieved by randomizing $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$ before the main loop of the scalar multiplication. The resulting TVLA trace is shown in the right of Figure 5.5, where we observe that there is no significant leakage with 1×10^6 traces. Applying this randomization, we can prevent from overwriting the registers with values having Hamming distance of zero.

5.6 Conclusion

In this chapter, we show how an exponent can be split into two shares, where the exponent is the XOR sum of the two shares and the cost is typically an extra register and some register copies per bit. This approach offers a significant advantage over previously proposed exponent splitting methods, which have a prohibitive impact on performance [CJ01]. Our method can also be applied to groups whose order contains long runs of bits set to 0 or 1 without any penalty on performance or security. We showed that our algorithms are secure using formal methods, MI-based evaluation and TVLA on an implementation of Boolean exponent splitting. Furthermore, we showed how an ECDSA signature can be generated by only manipulating shares of the random nonce used to secure the scheme. We note that our method does not prevent an attacker from using the intermediate states generated by the algorithms as a means of attack. However, inexpensive solutions such as randomizing projective points [WMPW98] or Ebeid et al.'s blinding method for RSA [EL10] can be combined with our method to provide a high level of side-channel resistance.

The algorithms presented above will be more efficient than adding a multiple of the group order to the exponent, since the bit length of the exponent is not increased. Moreover, the resistance to collision attacks is superior, since one would need to conduct several attacks to derive each share and reconstruct the exponent. Where one is adding a random multiple of the exponent any bits recovered directly relate to bits of the exponent used. It has not yet been shown that one can derive an exponent from gaining partial information on a series of blinded exponents, but significant advances have been made [SI11, JL12, Sch14, WS14].

Chapter 6

Residue Number System

> Wherever there is modularity there is the potential for misunderstanding: Hiding information implies a need to check communication.
>
> *Alan J. Perlis, Epigrams on Programming*

Residue Number System (RNS) is an arithmetic representation of an integer by its residues in a given base of coprime numbers. RNS is appropriate for parallelization of computations and it can speed up public-key cryptographic implementations. In this chapter, we present a secure RNS based Montgomery power ladder algorithm implemented on an ARM Cortex A8 processor. We perform a generic and thorough evaluation for various scalar multiplication implementations with RNS and traditional Side-Channel Attack (SCA) countermeasures, in an effort to assess the resistance of RNS against SCA. Three different countermeasures, namely scalar and point randomization and random base permutations, are implemented and evaluated using Test Vector Leakage Assessment (TVLA) and template attacks. More specifically, variations of RNS-based Montgomery power ladder scalar multiplication algorithms are evaluated on a BeagleBone Black using an EM probe for acquisition of the traces. We show experimentally and theoretically that new bounds should be put forward when TVLA evaluations on public key algorithms are performed. On the security of RNS, our data and location dependent template attacks show that even protected implementations are vulnerable to these attacks. A combination of RNS-based and traditional countermeasures is the best way to protect against side-channel leakage.

6.1 Introduction

Residue Number System (RNS) arithmetic is gaining grounds in public key cryptography, because it offers fast, efficient and secure implementations over large prime

fields or rings of integers. In cryptographic applications of public key cryptography, the recommended bit length varies between 256 bits (for ECC) to 3072 bits (for RSA) [Gir]. RNS-based implementations offer the possibility to perform the required modular operations in smaller numbers by distributing the integer operations on the residue values. Furthermore, RNS computations can be executed independently for each modulo, allowing parallelization of the calculations.

A broad range of countermeasures has been devised, in order to protect cryptographic operations against SCA. In Elliptic Curve Cryptography (ECC) where scalar multiplication is the key security operation, traditional countermeasures focus on randomizing the scalar, randomizing the input point or manipulating the EC parameters and point representation (e.g. randomized projective coordinates). An extensive study of countermeasures proposed for ECC is done by Fan and Verbauwhede [FV12a]. Several research groups explore alternative SCA resistance approaches that are focused on redundant arithmetic systems like the Nonadjacent Form (NAF) and the Residue Number System (RNS).

RNS was originally devised for parallel processing of arithmetic operations in order to increase computation speed [PP95, BDK97]. Since it can effectively represent elements of cyclic groups or finite fields there is merit in adopting it in finite field operations for elliptic curve arithmetic. In 1997 Bajard et al. [BDK97] presented an RNS Montgomery modular multiplication for very large operands by adapting the Montgomery multiplication to the mixed radix system. Their algorithm can be implemented to run in $\mathcal{O}(n)$ time on $\mathcal{O}(n)$ processors, where n is the number of moduli in the RNS basis. Efficient RNS hardware implementations of Montgomery multiplication for elliptic curves [BDE10] and RSA [CNPQ04] showed the potential of using RNS for public key applications. Furthermore, RNS has been recently used for other cryptographic schemes such as Lattice-based cryptography [BEHZ16] or in the work of Halevi et al. [HPS18], in order to optimize the implementation of the Fan-Vercauteren variant of the scale-invariant homomorphic encryption scheme of Brakerski [FV12b]. As this number system has a broad potential for cryptography, using it for SCA resistance seems to be inevitable. However, a practical evaluation of RNS as an SCA countermeasure is still not performed.

6.1.1 Related Work

The potential of RNS as an SCA countermeasure is observed in several research papers, for instance Bajard et al. in [BILT04, BEG13], Guillermin [Gui11], Fournaris et al. [FKSK15, FPBS16], Schinianakis et al. [SS13]. RNS parallel processing of finite field operations apart from speed offers also different representation of the elliptic curve points, which may reduce SCA leakage. Also, a single bit fault in an RNS number's moduli can lead to “difficult to trace" changes in an overall finite field element, supports the argument that the RNS can be introduced as SCA and Fault injection Attack (FA) countermeasure. It has also been observed that the periodic

change using base permutation during the modular exponentiation (and consecutively scalar multiplication) computation flow can introduce enough randomness to thwart SCAs. This approach led to the introduction of the leak resistant arithmetic (LRA) technique [BILT04]. LRA has been applied to modular exponentiation designs in two ways, either by choosing a new base permutation once at the beginning of each modular exponentiation or by choosing a permutation once in each modular multiplication during exponentiation [FPBS16].

Theoretical evaluations for secure RNS scalar multiplication are presented in [FPS17,CATB18]. Both papers discuss the resistance of RNS randomization against various SCA attacks. In [FPS17] we collected traces for an initial analysis on a Raspberry Pi 2, but due to the constrains of the target platform, no proper evaluation was made. The SCA analysis of the [FPS17] implementation is limited to identifying the rounds of RNS with and without the countermeasures. We imply that an Online Template type of attack (OTA) [BCP+14] should not be possible in our protected implementation, but detailed SCA results are missing. In [CATB18] Monte Carlo simulations are used to verify the resistance of the countermeasure against several attacks, but no practical t-tests are made. To the best of our knowledge, the only practical evaluation of an RNS hardware implementation is presented by Perin et al. [PITM13] but it is constrained to an RSA design and is focused on comparative horizontal attacks (Doubling/relative Doubling attacks) and signal to noise measurement estimations for SCA resistance.

6.1.2 Our Contribution

The above observations highlight the need for a practical, broad and generic evaluation approach of scalar multiplication implementations that rely on RNS arithmetic and led us to the results of this chapter. Such an approach and overall assessment could provide a definite answer on if and how RNS and its LRA technique can contribute to a scalar multiplier overall SCA resistance. This answer is based not only on theoretical analysis but, most importantly, on actual measurements.

In an effort to formulate a generic and broad methodology for evaluating the security of an RNS scalar multiplier, we use Test Vector Leakage Assessment (TVLA), initially proposed by Goodwill et al. [GJJR11] and presented in Chapter 3.6. This methodology consists of several statistical tests between two trace sets of acquisition and uses Welch's t-test to evaluate if the two sets have significant statistical differences, that would distinguish for example a fixed versus a random input. This test provides results simultaneously for all the intermediate values and indicates potential points of leakage. Since its introduction to public-key algorithms in 2011 [JRW11], few research groups have used TVLA so far for the evaluation of RSA or ECC. Namely, Nascimento et al. [NLD15], Chmielewski et al. [CMV+17] used it for evaluation of Curve25519 on Chipwhisperer and the complete Weierstrass formulas on an FPGA respectively. Tunstall and Goodwill [TG16] give an

overview of cases and algorithms that can be applied during public key TVLA evaluations. TVLA requires thousands or even millions of acquired traces to show the existence of leakage. This fact, combined with the low operation speed and the large trace lengths of public key implementations, make TVLA evaluations quite challenging for public-key cryptography. In Section 6.3, we show how to apply TVLA in an even more challenging target implementation, a RNS software implementation for the Montgomery power ladder on ECC. Theoretical analysis of TVLA for ECC leads to new threshold settings, which are validated by our experiments.

Furthermore, we developed a complementary template attack methodology, using data and location dependent leakage. Our goal is to validate the TVLA results and to expose additional vulnerabilities in the protected RNS designs. The template attacks are mounted on RNS LRA protected scalar multiplication and, indeed, achieve high success rate results in classifying correctly the template traces. *Location dependent template attacks* are introduced by Heyszl et al. [HMH+12] and used to attack an ECC implementation on an FPGA. We use here location dependent leakage templates for the first time in an RNS-ECC implementation setting.

6.1.3 Organization of this Chapter

This chapter includes the results of the author's research regarding RNS as a countermeasure in ECC implementations. More precisely, the combined results of three papers are presented. After briefly introducing the potential of using RNS as an SCA and FA countermeasure following [FPBS16], Section 6.2 highlights the basics of RNS arithmetic and presents the implementation of a secure, blinded Montgomery power ladder algorithm with RNS as published in [FPS17]. Then, the SCA resistance of this algorithm is discussed together with some preliminary results. In Section 6.3 we present a thorough security analysis of this algorithm using TVLA and template attacks. The implementation runs on a Cortex A8 processor and the experiments are performed for protected and unprotected versions of the implementation. The countermeasures tested include the typical randomization of the input point and an RNS specific technique based on randomization of the bases. Finally, in Section 6.4 we perform template attacks (data and location dependent) on the different variations of the implementation, in order to examine the behavior of the countermeasures. All these results will be published in the paper [PFPB19].

6.2 Efficient and Secure RNS Software Implementation for ECC

6.2.1 RNS Arithmetic for ECC

RNS is a non-positional arithmetic system where an integer X is represented by a set of individual n moduli x_i ($x \overset{RNS}{\rightarrow} X : (x_1, x_2, ...x_n)$) of a given RNS basis $B : (m_1, m_2, ...m_n)$ as long as $0 \leq x < M$ where $M = \prod_{i=1}^{n} m_i$ is the RNS dynamic range and all m_i are pair-wise relatively prime. The elements of the RNS basis need to be coprime, in order to uniquely reconstruct X, as shown in the proof the Chinese Remainder Theorem (CRT).[1] We recall here that CRT states the following:

Theorem 1. We consider a n-tuple of coprime numbers $(m_1, m_2, \ldots, m_n)$. We note $M = \prod_{i=1}^{n} m_i$. If we consider the n-tuple $(x_1, x_2, \ldots, x_n)$ of integers such that $x_i < m_i$, then there exists a unique X, such that $0 \leq X < M$ and $x_i = X \pmod{m}_i = |X|_{m_i}$ for $1 \leq i \leq n$.

We can clearly see that RNS is based on the CRT. Each x_i in an RNS basis can be derived from X by calculating $x_i = \langle x \rangle_{m_i} = x \ mod \ m_i$. In contrast to binary arithmetic, addition, subtraction and multiplication in RNS is performed within each modulus (in n independent channels) thus enabling the parallelism of small bit length operations to obtain an arithmetic result [BDK01, FKSK15]. Assuming that we have two integers a and d represented in RNS as $A : (a_1, a_2, ...a_n)$ and $D : (d_1, d_2, ...d_n)$ we can obtain addition, subtraction and multiplication in RNS as $A \oslash D = (\langle a_1 \oslash d_1 \rangle_{m_1}, ... \langle a_n \oslash d_n \rangle_{m_n})$ where $\oslash : (+, -, \times)$. Exact division by D coprime with M is equivalent to multiplying by the inverse $\langle D^{-1} \rangle_M$. Since RNS is a non-positional representation, comparisons, divisions and modular reductions are complex operations, which are performed either by converting the number from RNS to binary representation or by using base extension algorithms.

Binary reconstruction from RNS representation can be done either by using the Chinese Remainder Theorem (CRT) or through a Mixed Radix System (MRS) transformation.

Using CRT, an integer x can be written as $x = \left\langle \sum_{i=1}^{n} \left\langle x_i \cdot M_i^{-1} \right\rangle_{m_i} \cdot M_i \right\rangle_M$ where $M_i = \frac{M}{m_i}$ and M_i^{-1} is the multiplicative inverse of M_i modulo m_i. Due to the high bit length of M , the modular inverse is computationally demanding; it is typically computed by introducing a correction factor w, where $x = \sum_{i=1}^{n} \left\langle x_i \cdot M_i^{-1} \right\rangle_{m_i} \cdot M_i - w \cdot M$ (introducing the concept of Cox and Rower [KKSS00]). Using the MRS approach, this correction factor can be avoided but RNS numbers need to be transformed into MRS representation (a weighted moduli

[1] The earliest known statement of the theorem is by the Chinese mathematician Sunzi in the 3rd century AD.

RNS variant) [BILT04] and then from this representation to binary numbers. This technique will be explained in the next section.

For elliptic curves defined over *GF(p)* (elliptic curves on $GF(2^k)$ are not discussed in this chapter), all *GF(p)* operations (addition, subtraction, multiplication) are modular operations. The RNS modular multiplication over *GF(p)* is a computationally difficult operation. It is usually realized through the RNS Montgomery multiplication algorithm that avoids modular inversions, but includes base extension operations [BDK97, FPBS16, Fou17].

6.2.2 RNS Base Extension

Base extension (BE) is used when an RNS number represented in an RNS base $B_n = (m_1, m_2, \ldots, m_n)$ needs to be represented in a different base $\acute{B_n} = (m_{n+1}, m_{n+2}, \ldots, m_{2n})$. The elements of bases should be coprime, that is $\gcd(m_i, m_j) = 1$ for all $i \in \{1, \ldots, n\}$ and $j \in \{n+1, \ldots, 2n\}$. Base extension can be realized in various ways, but essentially, it consists of one step where the RNS number is transformed to binary using base B_n and a second step where the binary number is transformed to RNS using base $\acute{B_n}$. As such, base extension is strongly related to the methods of transforming an RNS number to binary.

Two main approaches to base extension are used in practice for RNS arithmetic: the Mixed Radix System (MRS) and the Cox-Rower architecture introduced in [KKSS00]. The Cox-Rower architecture consists of parallel arithmetic units, the Rowers, which perform the independent computations for each base concurrently, and the Cox unit dedicated to the computation of an approximation of the correction factor w. Therefore, it can be efficiently implemented in hardware. An interesting work in protecting the Cox-Rower architecture against multi-fault attacks is presented in [BEM16]. While the Cox-Rower method favors parallelism, the MRS system is often used for RNS Montgomery Multiplication base extension, because it offers benefits in terms of leakage resistance and fault propagation as discussed in [Fou17, BILT04, BEG13]. Moreover, MRS can be combined with techniques, such as the Leak Resistant Arithmetic, described in Section 6.2.3 and it can be efficiently implemented in software. The representation of an integer x in MRS is $\widetilde{X} : (u_1, u_2, ..., u_n)$; the MRS can be used for RNS to binary conversion. Using the naive approach of [ST67], the MRS number $\widetilde{X}$ can be obtained from $X : (x_1, x_2, ..., x_n)$ by executing the Mixed Radix Conversion (MRC) algorithm of Equation 6.1.

$$
\begin{aligned}
u_1 &= x_1 \\
u_2 &= \left\langle (x_2 - u_1) \cdot m_{1,2}^{-1} \right\rangle_{m_2} \\
u_3 &= \left\langle ((x_3 - u_1) \cdot m_{1,3}^{-1} - u_2) \cdot m_{2,3}^{-1} \right\rangle_{m_3} \\
&\dots \\
u_n &= \langle ((x_n - u_1) \cdot m_{1,n}^{-1} - u_2) \cdot m_{2,n}^{-1} - \dots - u_{n-1}) \cdot m_{n-1,n}^{-1} \rangle_{m_n} .
\end{aligned}
\tag{6.1}
$$

The multiplicative inverse of m_i modulo m_j is $m_{i,j}^{-1}$, i.e. $m_i \cdot m_{i,j}^{-1} \equiv 1 \; mod \; m_j$. From the MRS number representation, an integer x can be recovered by performing $x = u_1 + g_2u_2 + g_3u_3 + \ldots + g_nu_n$ where $g_i = \prod_{j=1}^{i} m_j$. In [BMP05], the authors observe that the conversion from binary to MRS can be reduced to one shift and four additions, then conversion to binary needs at most two shifts and four additions. As an example with a three elements base, $\{m_1, m_2, m_3\} = \{2^k - 1, 2^k, 2^k + 1\}$, we see that $m_1 \pmod m)_2 = 2, m_1 \pmod m)_3 = 1$ and $m_2 \pmod m)_3 = -1$. For the elements needed in Equation 6.1, $m_{1,2}^{-1} = 2^{n-1}, m_{1,3}^{-1} = 1, m_{2,3}^{-1} = -1$, thus one shift and one bitwise addition is necessary for the calculation of u_1 and three additions for u_2, u_3.

In MRS base extension, the integer X_{B_n} is converted into a number in base $\acute{B}_n$ $X_{\acute{B}_n}$ and vice versa. A similar two step procedure is followed for base extension from $\acute{B}_n$ to B_n. As we showed in the previous example, an efficient basis extension operation relies heavily on the choice of B_n and $\acute{B}_n$. Most studies on optimal base moduli agree that moduli of the form of Mersenne numbers $2^k \pm c_i$, $2^k - 2^{t_i} \pm 1$ or 2^k, $2^k - 1$, $2^{k-1} - 1$, $2^{k+1} - 1$ for various values i, n, k, provide good performance results [BKP09, ESJ+13]. This happens because division in modular reduction can be replaced with bitwise shifts and `AND` operations. Recent results from Bigou and Tisserand in [BT15] show how to perform RNS modular multiplication with a single base bit width instead of a double one, which results in two times faster implementation for the same area, but in this case the Leakage Resistant Arithmetic (LRA) approach followed in our approach cannot be used.

6.2.3 Leak Resistant Arithmetic

In 2004, Bajard et al. proposed, originally for modular exponentiation, a random permutation of the base B_n and $\acute{B}_n$ moduli thus creating $\binom{2n}{n}$ random permutations of B_n and $\acute{B}_n$ [BILT04]. We denote each such RNS Base γ permutation as $B_{n,\gamma}$ and $\acute{B}_{n,\gamma}$. The periodic change of a base permutation during the modular exponentiation, as presented in Figure 6.1, can introduce enough randomness to thwart SCAs as long as the number of moduli is high. This approach leads to a leak resistant arithmetic (LRA) technique that can be applied to modular exponentiation designs

(used for RSA) in two ways, either by choosing a new base permutation once at the beginning of each modular exponentiation or by changing a permutation in each RNS Montgomery Multiplication (RNS-MM) operation of the exponentiation process. The base transition of an RNS number A represented in a base permutation γ to a new permutation $\acute{\gamma}$ can be achieved by performing two consecutive RNS-MMs. Initially $A_1 = RNS - MM(A, M_{B \cup \acute{B}} \ mod \ P, P, \acute{B}_{n,\acute{\gamma}}, B_{n,\acute{\gamma}})$ [2] is performed and it is followed by $RNS - MM(A_1, 1, P, \acute{B_{n,\gamma}}, B_{n,\gamma})$

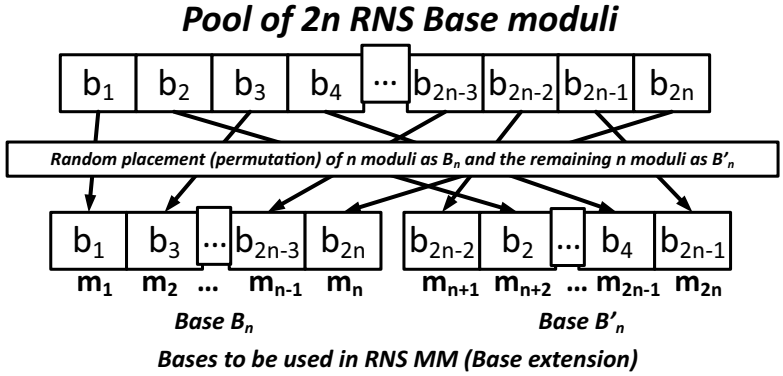

Figure (6.1) Leak Resistant Arithmetic approach to Base Randomization

Applying the LRA technique in scalar multiplication follows a similar approach to modular exponentiation. Some attempts to introduce LRA in scalar multiplication have been made in [Gui10, Gui11], however, they are applicable only to the CRT type base extension using the Cow-Rower method when pseudo-Mersenne numbers are used for base moduli. In scalar multiplication, a permutation transition can be done

1. only once per scalar multiplication,
2. in every round of the scalar multiplication process,
3. before every *GF(p)* RNS-MM operation of every point operation of every round.

Taking into account that the transition from one permutation to another costs 2 RNS-MM, the third approach is not affordable in terms of speed. As suggested in [FPS17], the first approach may be vulnerable to horizontal SCA attacks (depending on the implementation methodology). Therefore, the second approach where the RNS bases are permuted once per scalar multiplication round, is the most promising, offering a balance between performance and SCA resistance strength.

[2] Note that A has the form $A \cdot M_{B_{n,\gamma}} \ mod \ P$ (Montgomery form) since it is an output of some previous RNS-MM.

6.2.4 RNS Modular Multiplication

As mentioned in Section 6.2.1, the modular multiplication in RNS format is a computationally expensive operation. It is usually realized by Montgomery modular multiplication for efficiency reasons.

The Montgomery multiplication is a modular multiplication where one reduction is performed at each iteration of the multiplication. When the operands are in Montgomery form, the reduction can be performed by a shift instead of a division. This is a challenging property to use in RNS, since it is a non-positional representation. However, Bajard et al. presented in [BDK97] a RNS-MM for very large operands based on an adaptation of the Montgomery methods to mixed radix systems. The basic idea of the algorithm is that the least significant digit of a mixed radix representation can be chosen as any one of the residues of the RNS representation when the two systems are based on the same set of moduli. A simplified variation of this algorithm is presented in Algorithm 27.

Assuming that we introduce two RNS bases $B_n = (m_1, m_2, \ldots, m_n)$ and $\acute{B_n} = (m_{n+1}, m_{n+2}, \ldots, m_{2n})$ such that $\gcd(m_i, m_j) = 1$ for all $i \in \{1, \ldots, n\}$ and $j \in \{n+1, \ldots, 2n\}$, we express a *GF(p)* number x in base B_n or $\acute{B_n}$ as X_B and $X_{\acute{B}}$ respectively, while in both RNS bases as $X_{B \cup \acute{B}}$. We also define $M_B = \prod_{i=1}^{n} m_i$ and M_B^{-1} as the multiplicative inverse of M_B in base B_n, as well as $M_{\acute{B}} = \prod_{i=n+1}^{2n} m_i$ and $M_{\acute{B}}^{-1}$ as the multiplicative inverse of $M_{\acute{B}}$ in base $\acute{B_n}$. The RNS Montgomery multiplication (RNS-MM) as an outcome calculates $S_B = A \cdot B \cdot M_B^{-1} \mod p$ and $S_{\acute{B}} = A \cdot B \cdot M_{\acute{B}}^{-1} \mod p$. Base extension from one base to the other in RNS-MM is needed, since M_B^{-1} does not exist in base B_n and therefore computations must be migrated to the $\acute{B_n}$ base to come up with S_B.

In the first step of RNS-MM Base extension operation, the base B_n RNS number is converted into a base B_n MRS number. In the second step, the base B_n MRS number is converted into a base $\acute{B_n}$ RNS number. A similar two step procedure is followed for base extension from $\acute{B_n}$ to B_n respectively to provide a correct RNS-MM outcome.

It must be noted that each RNS number A used in Montgomery multiplication must be represented in the Montgomery format, meaning in the form $A_B \cdot M_B \; mod P_B$ or $A_{\acute{B}} \cdot M_{\acute{B}} \; mod P_{\acute{B}}$. To transform a number in the Montgomery normalized form, an RNS-MM must be performed between A and $M_{B \cup \acute{B}} \; mod \; P$ using the bases B_n and $\acute{B_n}$ in reverse order (i.e. RNS-MM$(A, M_{B \cup \acute{B}} \; mod P, P, \acute{B_n}, B_n)$). To leave the Montgomery domain we must perform an RNS-MM of number A with 1 (i.e. RNS-MM$(A, 1, P, B_n, \acute{B_n})$).

To increase computation efficiency, most studies on optimal base moduli [BKP09] agree that moduli of the form $2^k \pm c_i$, $2^k - 2^{t_i} \pm 1$ or 2^k, $2^k - 1$, $2^{k-1} - 1$ $2^{k+1} - 1$ (Mersenne numbers) for various i values provide high performance results. As we showed earlier, if the elements of the bases are some power of 2, the division is

Algorithm 27: RNS Montgomery Modular Multiplication RNS-MM$(A, D, P, B_n, \acute{B_n})$

Input: $B_n = (m_1, \ldots, m_n), \acute{B_n} = (m_{n+1}, \ldots, m_{2n})$,
$P_{B\cup\acute{B}} = P_B \cup P_{\acute{B}} : (p_1, p_2, ...p_n, p_{n+1}, ...p_{2n})$,
$M_B = \prod_{i=1}^{n} m_i$, $M_{\acute{B}} = \prod_{i=n}^{2n} m_i$,
$A_{B\cup\acute{B}} = A_B \cup A_{\acute{B}} : \{a_1, \ldots, a_n, a_{n+1}, \ldots a_{2n}\}$,
$D_{B\cup\acute{B}} = D_B \cup D_{\acute{B}} : \{d_1, \ldots, d_n, d_{n+1}, \ldots, d_{2n}\}, M_{\acute{B}}^{-1}, -P_B^{-1}$
Output: $S_B = A_B \cdot D_B \cdot M_B^{-1}\ mod P_B$ and
$S_{\acute{B}} = A_{\acute{B}} \cdot D_{\acute{B}} \cdot M_{\acute{B}}^{-1}\ mod P_{\acute{B}}$

1 $G_{B\cup\acute{B}} = A_{B\cup\acute{B}} \times D_{B\cup\acute{B}}$
2 i.e. $(g_i = \langle a_i \times d_i \rangle_{m_i}$ in base B_n and $\acute{g_i} = \left\langle \acute{a_i} \times \acute{d_i} \right\rangle_{\acute{m_i}}$ in base $\acute{B_n})$
3 $Q_B = G_B \times (-P_B^{-1})$ i.e. $\left(\left\langle g_i \times \left\langle -p^{-1} \right\rangle_{m_i} \right\rangle_{m_i} \right)$ in B_n
4 $Q_B \rightarrow Q_{\acute{B}}$ Base extension $B_n \rightarrow \acute{B_n}$;
5 $R_{\acute{B}} = G_{\acute{B}} + Q_{\acute{B}} \times P_{\acute{B}}$;
6 $S_{\acute{B}} = R_{\acute{B}} \times M_{\acute{B}}^{-1}$;
7 $S_{\acute{B}} \rightarrow S_B$ Base extension $\acute{B_n} \rightarrow B_n$;
8 **return** *S_B and $S_{\acute{B}}$*

avoided in modular reduction and replaced by bitwise AND and shift.

The number of base moduli n should also be optimized as well as each moduli's k value (defining the bit length of all the values). Usually, such numbers are specified according to the *GF(p)* defining the EC. For RNS applications on ECC, RNS moduli are proposed with variant bit lengths, in order to achieve different dynamic ranges for implementations of 192, 256 and 320 bits. [3] The dynamic range of the RNS bases B_n and $\acute{B_n}$ must be close to the prime p ($4p < M$) specifying the underlying field arithmetic. The choice of moduli in the RNS bases does not depend on the type of curve, but on the prime field that the curve is defined on. Following the approach of [BKP09] and [ESJ+13] we adopt 4 moduli RNS bases ($n = 4$) of the form $B_n : (2^k - 2^{t_1} - 1, 2^k - 2^{t_2} - 1, 2^k - 2^{t_3} - 1, 2^k - 2^{t_4} - 1)$ and $\acute{B_n} : (2^k, 2^k - 1, 2^{k+1} - 1, 2^{k-1} - 1)$. In that case, we employ RNS bases that have a 200-bit dynamic range consisting of four approximately 50-bit moduli ($k = 50$) for each involved RNS base B_n and $\acute{B_n}$ as follows:

The above base moduli can simplify modular reduction during a *GF(p)* multiplication and provide simple multiplicative inverses $m_{i,j}^{-1}$ i.e. $m_{5,6}^{-1} = 1, m_{5,7}^{-1} =$

[3] The dynamic range of a base is defined as the difference between the largest and smallest element of the base. A large dynamic range offers higher levels of security due to the entropy produced by the variations of the Hamming weight of the base elements.

Table (6.1) Bases for *GF(p)* RNS Montgomery Modular Multiplication with p 192-bit prime

B_n	$m_1 = 2^{50} - 2^{20} - 1$	$m_5 = 2^{50}$
	$m_4 = 2^{50} - 2^{10} - 1$	$m_6 = 2^{51} - 1$
$\acute{B_n}$	$m_3 = 2^{50} - 2^{18} - 1$	$m_7 = 2^{51} - 1$
	$m_4 = 2^{50} - 2^{10} - 1$	$m_8 = 2^{49} - 1$

$2, m_{6,7}^{-1} = -2$. If the bases share a common m_i, as it is the case for Table 6.1, we could achieve more efficient implementations, since some values are computed faster. The security of such an implementation is not threatened, as long as the dynamic range of the bases is large. However, if an attacker can trigger the algorithm at this low arithmetic level, he could find similarities in the power signal and conclude that the same m_i is used.

6.2.5 RNS Montgomery Power Ladder Implementation

Considering that LRA can be a strong randomization tool in an SCA resistant scalar multiplication algorithm, we adopt a Montgomery powering ladder (MPL) algorithm for scalar multiplication that is proposed and analyzed in [FPBS16, FPS17]. This algorithm realizes LRA as a random RNS Base permutation once per scalar multiplication round and it is combined with more traditional techniques like base point randomization, in order to provide resistance to a broad range of SCA.

In the rest of this chapter, we extend the approach followed in [FPS17] and present the implementation of several RNS scalar multiplication algorithms. Furthermore, we provide experimental results on the SCA resistance of the LRA technique when used autonomously, with RNS random operation sequence or in combination with traditional SCA techniques. To achieve that, using the MPL algorithm publicly available in [Fou18b] as a starting point, we implemented four different scalar multiplier variants in C software code for embedded devices using the GMP 6.1.0 library. The GMP library was chosen for its usability and speed on calculating big-number values during scalar multiplication. Versions of GMP newer than the 6.0.0 release have side-channel resistant functions, such as silent modular operations without branching and constant time. Nevertheless, use of GMP during scalar multiplication is limited to basic RNS *GF(p)* building blocks (modular addition, subtraction) that are not directly related to sensitive information. We implemented our own big-number RNS-based Montgomery modular multiplication that is constant time. We also used the random generation function of GMP $mpz\_urandomb()$, which is a common software random generator. Therefore, we do not expect that this choice of library will affect the evaluation of our countermeasures in terms of side-channel leakage.

1. The first variant implements the original MPL algorithm [JY02] with no fur-

ther countermeasures.

2. The second algorithm implements an MPL optimization with base point randomization [Fou17, FV06], that is resistant against comparative, horizontal SCAs. This variant, presented in Algorithm 28 takes advantage of the MPL intrinsic coherence between two calculated points of each MPL round [Gir06] in order to create a fault detection mechanism [Fou17]. The correct result is unblinded at line 13, only if a fault is not induced.

3. In the third scalar multiplier implementation we infused LRA in the original MPL algorithm, that is one random base permutation conversion is performed per scalar multiplication round.

4. In the fourth variant we combined the countermeasures of variants (2) and (3). This is a protected algorithm with base point randomization and random base permutation and it is presented in Algorithm 29.

5. The fifth variant uses only RNS-specific countermeasures, namely the LRA technique described previously, and the random moduli sequence. Performing the operations using random moduli sequence offers a way of shuffling, which enhances the side-channel resistance of an RNS implementation as verified by our experiments.

We present the algorithms used in the second and fourth scalar multiplier implementations (Alg. 28 and Alg. 29 respectively). As it can be seen there is a need for a transformation to Montgomery format and a Random Base permutation (RBP) conversion for various points in the algorithm. According to [BILT04], RBP can be achieved by performing two consecutive RNS Montgomery multiplications per point coordinate that use the old and new permutations' bases B_n and $\acute{B}_n$ in reverse order. All four variants of scalar multiplication can use a fixed or a randomized scalar as input and are implemented with and without RNS random moduli operation sequence.

In all four implementations, *GF(p)* operations are performed in RNS. Performance optimizations are out of scope of this work. We aim to evaluate a straightforward implementation and any sort of optimization might influence our results, which is undesirable at this stage of evaluation. As is typically implemented in the literature and in order to avoid excessive operations unrelated to the scalar multiplication, values related only to RNS Bases moduli are precomputed and stored in memory for use in BE and random base permutation algorithms. Following the approach of [BKP09, ESJ+13] a four moduli RNS bases ($n = 4$) realization was used in all four RNS implementations. For all the above approaches, a *GF(p)* twisted Edwards EC was adopted with $a = 1$ and $d = 2$ where $p = 2^{192} - 2^{64} - 1$. Twisted Edwards Curves were chosen instead of Weierstrass ones since the first have never

Algorithm 28: Blinded SCA-FA MPL [Fou17]

Input: V, $R \in E(GF(p))$, $e = (e_{t-1}, e_{t-2}, ...e_0)$
Output: $e \cdot V$ or random value (in case of faults)

Choose base B_n and $\acute{B}_n$ permutation γ_t. ;
Transform V, R to RNS format using γ_t permutation;
$R_0 = R$, $R_1 = R + V$, $R_2 = -R$ in permutation γ_t;
Convert R_0,R_1,R_2 to Montgomery format;
for $i = t - 1$ **down to** 0 **do**
    $R_2 = 2R_2$, performed in permutation γ_t;
    if $e_i = 1$ **then**
        $R_0 = R_0 + R_1$ and $R_1 = 2R_1$ in permutation γ_t;
    end
    else
        $R_1 = R_0 + R_1$ and $R_0 = 2R_0$ in permutation γ_t;
    end
end
if *(i, e not modified and $R_0 + V = R_1$)* **then**
    return $R_0 + R_2$ *in perm.* γ_t
end
else
    return *random value*
end

been tested under the RNS arithmetic framework. However, the twisted Edwards curve shape was used and not the equivalent Montgomery form that can be combined with MPL, in order to keep the solution generic enough so as to be usable for other EC types. To retain compatibility with NIST Curves and implementations of other similar works [ESJ+13], the prime field was left to be $p = 2^{192} - 2^{64} - 1$ (although the implementation can be easily adapted to any Edwards Curve including the popular Curve 25519). The security of the chosen Edwards Curve does not alter the results of our leakage study, since it relies on the employed countermeasures. We performed experiments with the secure Edwards Curve (a=107, d=47, cofactor=4) as proposed in [BBJ+08] and came up with similar results that are presented in Section 6.3.5.

For *GF(p)* with p a 192-bit prime, we employ RNS bases that have a 200-bit dynamic range consisting of four approximately 50-bit moduli ($k = 50$) for each involved RNS base B_n and $\acute{B}_n$ as suggested in [BKP09, ESJ+13, FPS17]. Since $n = 4$ there exist 70 different base permutations, which as an individual SCA countermeasure, introduce a small randomization. The security level provided by 70 different bases might not seem enough, however this is a trade-off between memory

Algorithm 29: LRA SCA-FA Blinded MPL [Fou18b]

Input: V, $R \in E(GF(p))$, $e = (e_{t-1}, e_{t-2}, ...e_0)$
Output: $e \cdot V$ or random value (in case of faults)

Choose random initial base permutation γ_t. ;
Transform V, R to RNS format using γ_t permutation;
$R_0 = R$, $R_1 = R + V$, $R_2 = -R$;
Convert R_0,R_1,R_2 to Montgomery format
for $i = t - 1$ **down to** 0 **do**
    $R_2 = 2R_2$, performed in permutation γ_t ;
    Choose a random base permutation γ_i;
    RBP from γ_{i+1} to γ_i for R_0 and R_1 ;
    if $e_i = 1$ **then**
        $R_0 = R_0 + R_1$ and $R_1 = 2R_1$ in permutation γ_i;
    end
    else
        $R_1 = R_0 + R_1$ and $R_0 = 2R_0$ in permutation γ_i;
    end
    RBP from γ_i to γ_t for V;
end
if *(i, e not modified and $R_0 + V = R_1$)* **then**
    RBP from γ_0 to γ_t for R_0;
    return $R_0 + R_2$ *in permutation γ_t ;*
end
else
    return *random value*
end

cost and SCA resistance. Increasing the number of moduli would result in a big precomputation table (currently 70×70 50-bit numbers) used in every RNS base extension operation. Instead of increasing the number of moduli, we propose combining LRA with the low overhead technique of randomizing the RNS moduli operation sequence, offering 24 different combinations (for $n = 4$) for each RNS operation. By randomizing the sequence of the moduli operations, we take advantage of the basic RNS property of parallel computations, which offers a way of shuffling and enhances the side-channel resistance of an RNS implementation as verified by our experiments. This will enable us to create uniquely random computation patterns for each EC point operation and each MPL round.

6.3 Practical Evaluation of RNS Using Test Vector Leakage Assessment

As indicated in the above analysis, the emerging use of RNS systems for cryptographic implementations and the broad belief that RNS offers a potential SCA countermeasure makes practical evaluations of RNS implementations essential. The complexity of RNS together with the fact that software implementations of RNS are slow, make it challenging to apply common evaluation techniques for such systems. Considering the above challenges, we propose a broadly applicable practical evaluation approach based on TVLA and template attacks for RNS based scalar multiplication. Our primary goal is to examine and validate the RNS SCA capabilities using practical results. Our evaluation is performed on the RNS implementation described in Section 6.2.5, including both RNS and traditional SCA countermeasures.

6.3.1 New TVLA Threshold for Public Key Algorithms

The Test Vector Leakage Assessment (TVLA) methodology, initially proposed by Cryptography Research International (CRI) [GJJR11], is described in Section 3.6. As it is commonly used, we will apply several TVLA statistical tests as a first step towards evaluation of the SCA resistance of our RNS implementation with various countermeasures. In the proposed evaluation approach, we verify and extend previous theoretical results and simulations on the *boundaries of the t-test threshold* for the case of traces with high number of samples that are obtained by a public-key cryptographic implementation. A generic methodology to provide those boundaries is proposed.

Current TVLA evaluations for the Welch's t-test use the critical value of ± 4.5 [GJJR11, TG16, NLD15, CMV+17], which corresponds to a statistical significance level of $\alpha < 0.00001$ for the univariate test. However, in most previous works, the significance level of $a < 0.00001$ does not consider the total number of samples on the trace. As noted in [ZDD+17], the overall significance level increases as the

number of leakage points on the trace increases. Therefore, the authors propose to adjust the significance level according to the number of points on a trace. For long traces, meaning for more than 10^5 samples per trace, the overall test statistic value will be larger than ± 4.5 and therefore a non-leaky device can not pass the TVLA t-test with the critical value of ± 4.5. Hence, Balasch et al. [BGG+14] suggested raising the critical value to ± 5 for longer traces based on numerical experiments. For longer traces, as the ones obtained from the implementation of public-key algorithms, a non-leaky device can not pass the t-test even with this higher value of ± 5. However, numerical experiments for one implementation have no impact of what should be done for another implementation. A few peaks above ± 4.5 were observed also by the authors of [CMV+17], who used random versus random values, in order to check if those are ghost peaks and verify the security of their implementation. It is obvious that an explicit rigorous way of setting the threshold value would help for sound application of the current TVLA procedure in every implementation.

As mentioned in Section 3.6.1, one of the applications of Welch's t-test is to test the location of one sequence of independent and identically distributed random variables, as the ones obtained from SCA measurements. It is obvious that the number of errors per statistical test is relevant to the number of samples in the SCA traces. The Šidák correction $a_{SID} = 1-(1-a)^{n_s}$ provides a calibration of the significance level in relation to the number n of samples, and therefore changes the threshold of the statistical tests [DZD+17].

Taking all these into consideration we created a Matlab script which calculates the threshold value based on the number of samples and the variance of each sample in a given trace. We calculate the t-value according to the Šidák correction; the larger the number of traces obtained, the closer is the t-test value to the normal distribution's value. By using Algorithm 30 and 5 M samples per trace, we obtain a threshold of ± 6, which suggests that the boundaries can be relaxed further. This result is generic and should be applied, in order to define the threshold before any t-test evaluation starts. It makes more sense to use this algorithm when public-key cryptographic implementations are used or in any case that the traces contain more than 1 M samples, because of the false positives that would appear with the threshold set to ± 4.5

At this point, it is worth noticing that the above observation affects public key cryptographic TVLA evaluations and explains the results of previous papers. For instance, Nascimento et al. [NLD15] with 400 k samples and Chmielewski et al. [CMV+17] with 32M samples for their t-tests observed some peaks above ± 4.5, which they discarded as false positives. Using 400 k and 32 M samples in our Matlab script, with guessed values for standard deviations, gives threshold values of ± 6.7 and ± 7.3 respectively, which would evaluate their implementations as secure with no false positives. Therefore, it is important to adjust the boundaries for every set of traces.

Algorithm 30: t-test Threshold

Input: number of traces for group A and B nt_A, nt_B, number of samples n_s, sampled standard deviation s_A, s_B

Output: threshold value for student t distribution th_t

1 Choose level of significance a. Here $a = 0.00001$.;

2 Šidák correction $sidak_a = 1 - (1 - a)^{n_s}$;

3 $df = (\frac{s_A^2}{nt_A} + \frac{s_B^2}{nt_B})^2 / ...((\frac{s_A^2}{nt_A})^2 / (nt_A - 1) + ((\frac{s_B^2}{nt_B})^2 / (nt_B - 1)))$;

4 Threshold $th_t = tinv(1 - sidak_a/2, df)$

6.3.2 Experimental Setup

For the experiments we chose to load the implementation on a BeagleBone Black, which is a typical processor for portable devices. Apart from being a widely used processor, its high frequency of 1 GHz is necessary for our experiments, since a software implementation of RNS requires a lot of computational power to operate. As an indication, a full scalar multiplication for a 192-bit scalar takes 1.5 seconds when both SCA countermeasures are activated. The experimental setup we used, as depicted in Figure 6.2, is the following:

- BeagleBone Black with Cortex A8 processor running at 1 GHz.
- EMV Langer probe LF B-3, H Field 100 kHz up to 50 MHz.
- Lecroy Waverunner 8404M-MS sampling at 2.5 GS/sec.
- A Windows PC with the Inspector 4.12 and Matlab R2016b software.

Figure (6.2) The BeagleBone Black with the spot where the EMV probe is fixed to take measurements on the left and the complete setup with the Lecroy oscilloscope and the PC on the right

We used the secure RNS scalar multiplication algorithm from the public repository [Fou18b] modified with the appropriate functions to perform various types of t-tests and template attacks. For the analysis of our traces we used Matlab R2016b and the Inspector 4.12 software for Side-Channel Attacks provided by Riscure [Risa].

6.3.3 Processing of Traces and Alignment Technique

Misalignment of traces is a common obstacle in security evaluations. It is often due to the noise of the target device or it is imposed on purpose as "jitter", in order to make the target device more resistant to SCA. Having our implementation on the BeagleBone Black, where other processes of the operating system are running at the same time, it was expected to have some noise from the device. However, this is what makes a target and a leakage assessment more realistic. With some common signal processing techniques, such as the absolute value (ABS) operation, the window resampling and low pass filter, we could get a clear signal and we were able to perform TVLA. The misalignment due to random interrupts is handled by acquiring a big number of traces (in some cases we reached 50 k traces), and by throwing out the traces that had interrupts. These were not many, mostly it was about 1/10 of the total number of traces.

The ABS operation, as its name suggests, results in traces where the absolute value of each sample is depicted. The Inspector module of ABS computes the average value of each sample and uses it as a reference offset value. The result for each trace s_i in a trace set S is: Abs_offset$(s_i) = |s_i - \text{offset}|$.

Window resampling is a technique, where the samples of the acquired trace are resampled in a window of a desired length, in order to “clear out" some noise from the trace and make it shorter and easier to process. This technique offers a Signal-to-noise-ratio improvement by averaging several samples that carry the same signal. A second advantage is performance improvement. The compression of many samples into one, results in smaller traces which can be processed faster. As with every resampling technique, it results in some loss of information, which could possibly include leakage points and lower the leakage levels. This loss of information is prevented in our experiments by using a 75% of overlap of samples, at the cost of performance. After experimenting with different window values, 200 samples per window with an overlap of 75% gave good results, with minimal information loss and without affecting the leakage levels.

A technique that we applied to get clear patterns for alignment is the *Low Pass Filter*, which allows only certain frequencies to pass. From the Fast Fourier Transformation (FFT), we found the dominant frequencies of our device during execution of the cryptographic algorithms. As seen in Figure 6.3, the maximum energy segments are at $0 - 300$ MHz and a high frequency appears also at 1 GHz, the running frequency of our processor. As seen in Figure 6.4, applying a low-pass filter in a trace, would make patterns, as the selected one, more clear in all the traces. We

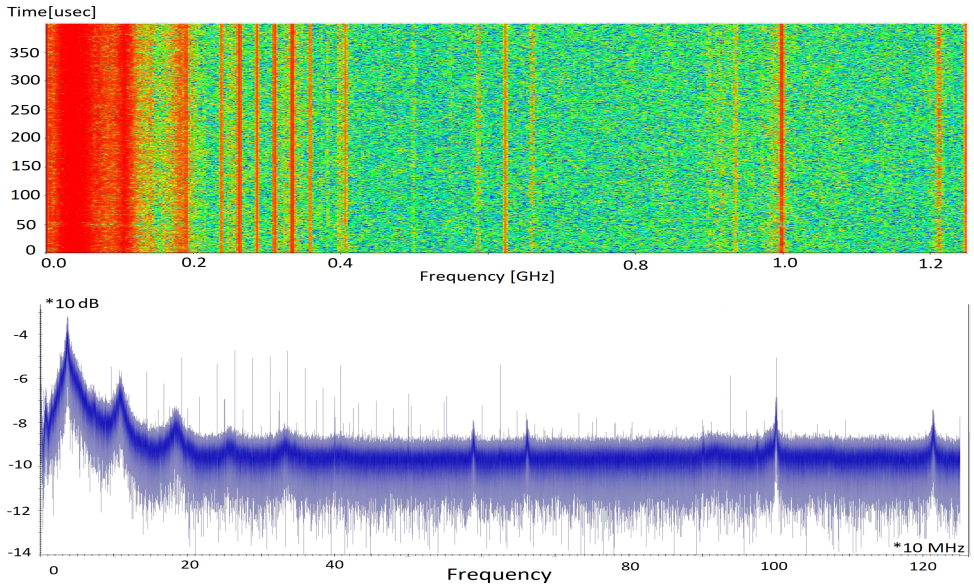

Figure (6.3) Applying FFT to find the dominant frequencies

chose these repetitive patterns to perform alignment.

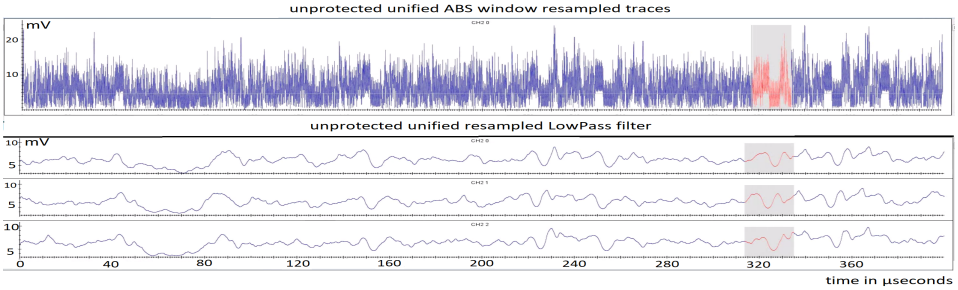

Figure (6.4) Applying low pass filter to find good alignment points

6.3.4 TVLA Analysis & Results

The leakage analysis for the RNS implementation was applied for the following four cases:

1. The unprotected version, where no countermeasures are incorporated apart from the fact that a constant time MPL implementation is used (function `unprotect()`).

2. Scalar multiplication with random input point (function `rdm_point()`).

3. Scalar multiplication with random base permutations (function `LRA()`).

4. The fully protected version of scalar multiplication, where both point and base permutation randomizations are applied (function `LRA_rdm_point()`).

We performed five sets of experiments. In the first experiment set, we performed t-tests for fixed versus random scalar, when the countermeasure of scalar randomization is not used. The scalar is usually the secret value of the protocols used in public key cryptography; for instance during scalar multiplication or during signing, the aim is to hide the scalar, i.e. the private key, from an adversary. Therefore, it is interesting to test the correlation of the implementation traces to random versus fixed scalar. In Section 6.3.4.1, we kept the secret scalar unmasked. In a second set of experiments, we differentiated between random and fixed input point. In Section 6.3.4.2, we present a series of t-tests for the four variations of the RNS implementation, in order to evaluate the correlation of the implementation leakage to a fixed or a random input point. Then, we implemented and tested scalar randomization. In the experiment set of Section 6.3.4.3, the t-tests for fixed versus random scalar show the correlation between the randomized scalar and the alternations between random and fixed values of it. Section 6.3.5 shows the same set of experiments using random versus fixed point and the implementation with unified formulas on the secure twisted Edwards curve. Finally, in Section 6.3.6 we implement two RNS specific countermeasures, namely the LRA with the RNS random moduli operation sequence, on the secure twisted Edwards curve. We then perform the t-tests using random versus fixed input points.

All sets of experiments that we performed, followed the rationale presented initially in [JRW11]. The acquired traces are compared with a constant scalar−constant input point test vector, in order to reveal any systematic relationship between the power consumption and the secret scalar or the input point respectively. The test fails if there is a single point in time where the t-value for the chosen set exceeds $+6$ or -6, the new t-test boundaries that we defined in Section 3.6.1 due to the large trace length of RNS scalar multiplication algorithm. The Matlab code used for the TVLA evaluation can be found in Appendix B.

6.3.4.1 t-test Random Versus Fixed Scalar

At first we performed the t-test in raw traces, without any pre-processing and alignment. Since we are interested in the leakage of the scalar, we performed the t-tests with fixed versus random scalar.[4] We observed that the levels of noise and misalignment were so high, that even in the unprotected case, the leakage was not significant. This is depicted in Figure 6.5, where we see that there is leakage in the beginning of

[4] At this point, we assume that we are able to control the input of the secret scalar, but in later experiments we have more realistic attack scenarios, where we can manipulate only the input points.

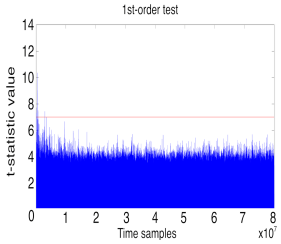

Figure (6.5) Not aligned traces

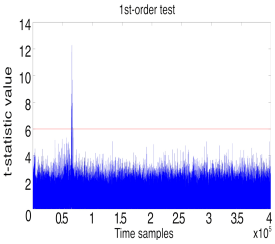

Figure (6.6) Alignment around 60000 samples

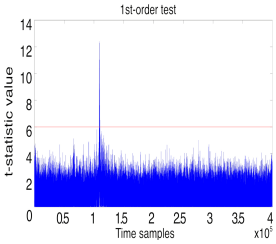

Figure (6.7) Alignment around 130000 samples

the traces, related to the processing of the scalar k. However these peaks disappear in the next rounds due to misalignment. In Figure 6.6, we notice a peak at the point where we performed alignment.

Apart from the fact that window resampling offers a natural way of alignment, we also performed static alignment in the resampled traces following the procedure described in Section 6.3.3. Aligning at interesting patterns rather than interesting points is useful in our case, since we have a large number of points in all experiments, ranging from 10^5 to $8 \cdot 10^5$.[5] A high density of leakage peaks is observed at the places where alignment is applied. It is noticeable that the leakage "moves" on the trace following the alignment points, as shown in Figures 6.6−6.7. This fact makes us conclude that the leakage is spread all over the trace and good alignment is necessary, in order to reveal it.

It is important to note at this point, that despite having two randomization countermeasures switched on, there is leakage from the unprotected scalar. This fact indicates the necessity to apply scalar randomization for a secure implementation, even if it might be a costly countermeasure. The following figures show the t-test results for all four cases for aligned traces obtained by interchanging between random and fixed scalars.

Figure 6.8 shows the leakage for the unprotected implementation where both countermeasures are switched off. Figures 6.9− 6.10 show the leakage when there is only the randomization of the input point or the random base permutation (LRA) respectively. In Figure 6.9 we can identify the manipulation of the scalar in every round and the results are in accordance with our expectations, as we have collected about 7 rounds of the implementation for the `rdm_point()` case. The first round is between $10^5 - 1.5 \cdot 10^5$ time samples, the second is between $1.510^5 - 2 \cdot 10^5$ time samples and so on. The peaks indicating the rounds are above the threshold line and they are the points where the highest leakage of each round is observed, due to a scalar-dependent operation. The initial peaks between $0 - 0.5 \cdot 10^5$ time samples

[5] When the number of samples is small, the method to find interesting points of Section 4.2 in [PITM13] can be followed.

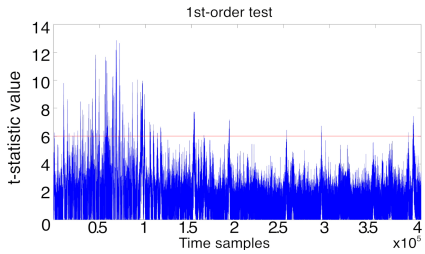

Figure (6.8) `unprotected()` random vs fixed scalar

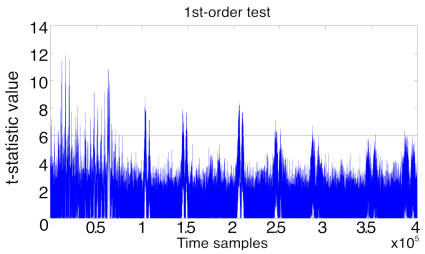

Figure (6.9) `rdm_point()` random vs fixed scalar

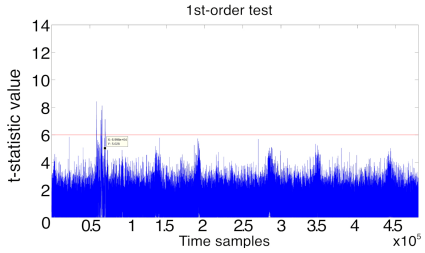

Figure (6.10) `LRA()` random vs fixed scalar

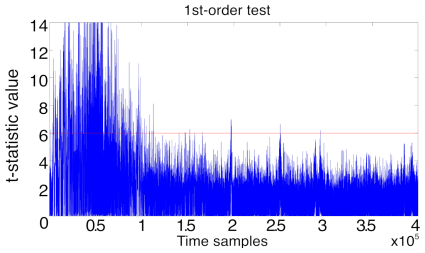

Figure (6.11) `LRA_rdm_point()` random vs fixed scalar

are also indicating leakage and they are associated with the initialization of the the variables, where also the value of the scalar is assigned to a variable.

An obvious observation is that the introduction of the LRA technique effectively reduces leakage compared to the t-test in Figure 6.10. Thus, it seems that the LRA technique is more effective as a countermeasure when compared to input point randomization. Finally, Figure 6.11 shows the leakage of the implementation with combined countermeasures. It is important to note here that the leakage of the scalar happens in the beginning of the execution, where most probably the location of the scalar is accessed and the value of the scalar retrieved, in order to get its most significant bits and start the bitwise scalar multiplication. This leakage is expected, since no scalar randomization is implemented in this case. It is also expected to see the leakage disappear after a few rounds, because the other two countermeasures are active and they are able to hide the scalar as well. What is not expected is that the combination of randomized base permutation (LRA) with randomized input point increases leakage, especially in the first MPL rounds. This fact might be platform specific and it seems to be possible to launch an attack during the first two MPL rounds of this implementation.

6.3.4.2 t-test Random Versus Fixed Input Point

As a next set of experiments, we take a set of randomly interleaved acquisitions of fixed versus random input point. The scalar is fixed and the countermeasures of randomized input point and/or random base permutations are applied as before. It is obvious for the t-test illustrations that the leakage in this case is significantly smaller than in Section 6.3.4.1.

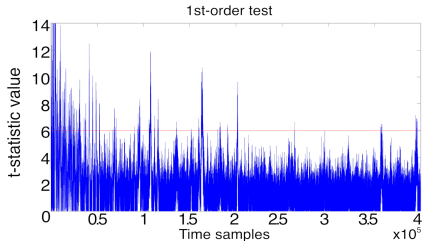

Figure (6.12) `unprotected()` random vs fixed input point

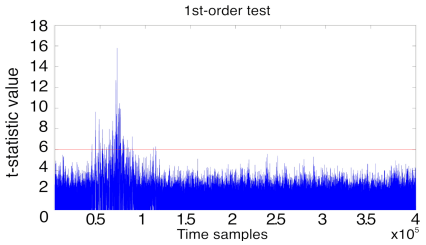

Figure (6.13) `rdm_point()` random vs fixed input point

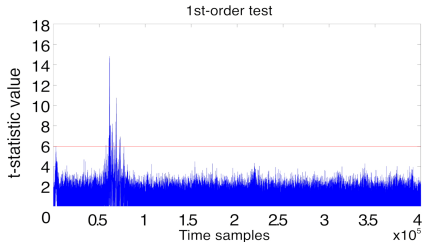

Figure (6.14) `LRA()` random vs fixed input point

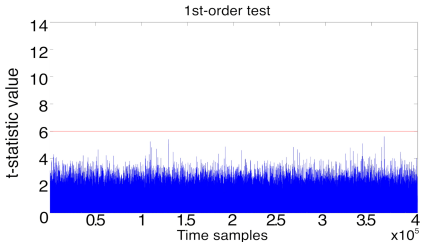

Figure (6.15) `LRA_rdm_point()` random vs fixed input point

In Figure 6.12 we see that there is significant leakage in the first rounds that reduces later, but this is only due to misalignment caused by the noise. When we set the trigger at a later round, starting for instance at the 10th scalar bit, we got a similar picture, with many leakage peaks in the beginning of the t-test. Figures 6.13−6.14 show that applying one of the two proposed countermeasures is enough to reduce the leakage of the implementation only around the aligned area, which might be enough to exploit the implementation, but due to the limited number of leakage points it would be hard to perform a successful attack. Finally, Figure 6.15 shows that our protected implementation passes the t-test as expected.

6.3.4.3 t-test Random Versus Fixed Scalar for Randomized Scalar Variation

The evaluation of a countermeasure with TVLA consists of distinguishing between random and fixed values of a certain parameter. If this parameter is randomized and

used as countermeasure, then an adversary should not be able to distinguish between a random or a fixed value of it. This is the case for this part of our experiments. t-tests for fixed versus random scalar, when the scalar randomization countermeasure is applied, show that there is no leakage in all four cases as expected. The same pre-processing steps of absolute value, window resampling and alignment are applied as before.

Figure 6.16 indicates the t-test results for the scalar multiplication variant of MPL with RNS without input point randomization and LRA, only the scalar randomization is applied. Since we perform 1st order t-tests and we blind the secret scalar, our implementation passes the statistical tests. Depending on the order of the masked value and the order of the t-test, it can happen that higher order t-test will fail. For instance, if additive splitting of the scalar with more than 2 shares is used, then the 1st order t-test should fail. For this work, we performed only 1st order t-tests and we blind the scalar by adding to it a random multiple of the order of the group. As indicated in all figures applying randomized scalar combined with the other RNS countermeasures can give a secure implementation.

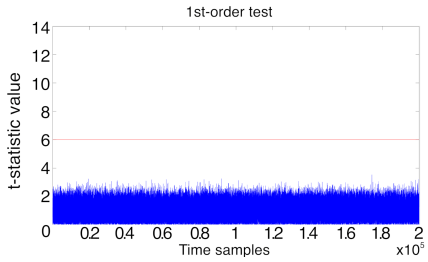

Figure (6.16) random vs fixed scalar`unprotected_scalar_rdm()`

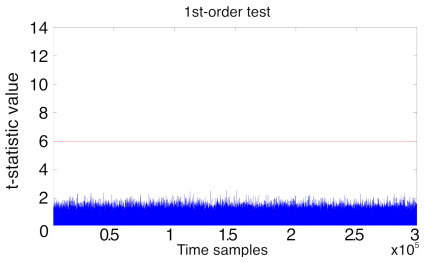

Figure (6.17) random vs fixed scalar `rdm_point_scalar_rdm()`

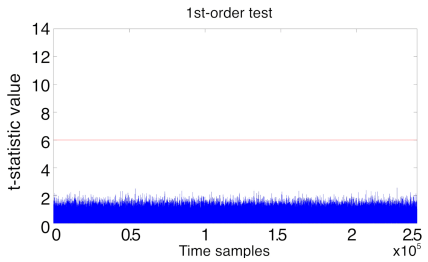

Figure (6.18) random vs fixed scalar `scalar_rdm_LRA()`

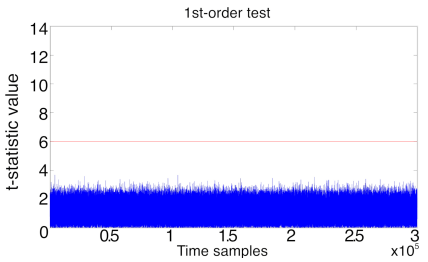

Figure (6.19) random vs fixed scalar `scalar_rdm_LRA_rdm_point()`

6.3.5 Secure Twisted Edwards Curve and Unified Formulas

The TVLA results presented above are independent of the chosen curve and the applied group law. The results of Sections 6.3.4.1- 6.3.4.3 are performed on the Edwards curve $a = 1, d = 2$, which is not included in the list of secure curves according to [BCLN16]. The tests of this section are performed on the secure twisted Edwards Curve $(a = 102, d = 47)$ [BBJ+08] and the results are similar to the previous ones. We also replaced the dedicated group law (which was chosen for efficiency reasons in the first series of experiments), with the unified group law.

The Figures 6.20- 6.23 show the t-tests for the case of random versus fixed input point. As we see from these figures, the number of samples is of order 10^4, instead of 10^5 as it was in previous experiments. Higher misalignment levels in the new traces compared to previous ones, led us to perform the processing steps with different parameters. Apart from that, the graphs are similar to that of Section 6.3.4.2, with much leakage for the unprotected version, less leakage when one countermeasure is applied, and no leakage for the fully protected version. In Figure 6.22, we see that applying only randomization of the point cannot hide the correlation between random and fixed points, and this leakage is spread throughout the trace, not only in the aligned area as before. This might be due to the fact that we achieved better alignment in this set of traces, so the leakage is more obvious.

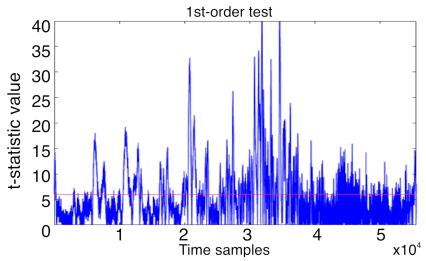

Figure (6.20) rdm vs fixed point `unprotected()`

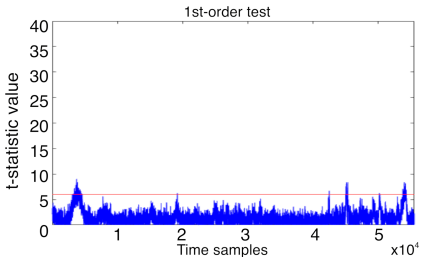

Figure (6.21) rdm vs fixed point `LRA()`

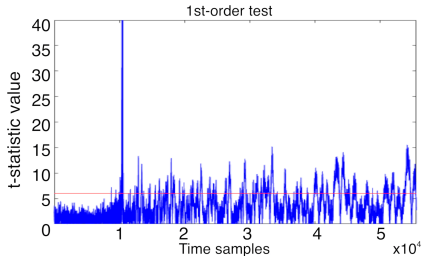

Figure (6.22) rdm vs fixed point `rdm_point()`

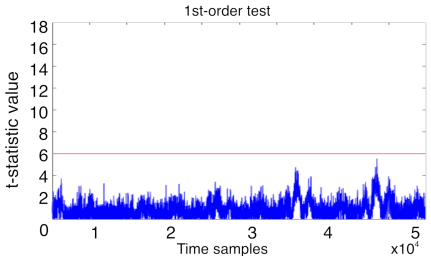

Figure (6.23) rdm vs fixed point `LRA_rdm_point()`

6.3.6 Secure Twisted Edwards Curve with RNS Random Moduli Operation Sequence

At this point, we examine the TVLA results of two RNS-specific countermeasures, namely the LRA technique and the RNS random moduli operation sequence technique. We notice that the leakage is significantly less compared to the results of the previous sections for the case of LRA. This is due to the fact that extra randomization is added in the implementation.

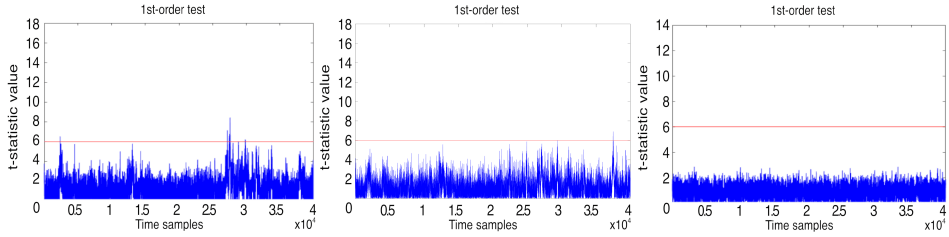

Figure (6.24) t-tests for random moduli operation sequence `LRA_rdm_oper()`

Figure 6.24 (left) shows the results for random versus fixed scalar, when the scalar is not randomized. Leakage is observed as expected, but it is much less compared to Figure 6.10 from previous results. Figure 6.24 (middle) shows the results for random versus fixed scalar, when the scalar is randomized. We notice only one peak around the 38000 sample, which is above the threshold and can be considered a ghost peak, since it disappears when we perform random vs random TVLA. Finally, Figure 6.24 (right) shows the results for random versus fixed point, when the point is random and shows clearly no leakage. This is an improvement over previous results for LRA in Section 6.3.4.2. The above results imply that the combination of the two RNS countermeasures lead to reduced leakage compared to implementations with single countermeasures or LRA with base point randomization.

6.4 Template Attacks on RNS Scalar Multiplication

TVLA offers a generic framework for evaluating an implementation and the t-tests presented in the previous section show the expected leakage of RNS with the combination of LRA and traditional countermeasures. However, there are several issues with TVLA about coverage of all possible values: how can we be sure that the fixed scalar "triggers" all parts of the circuit to ensure that any possible leakage that could occur does actually occur? TVLA offers an indication about the effectiveness of the applied countermeasures, but failing to mount an actual attack shows strong evidence for that. In this section, we perform template attacks in the different variations of the implementation, in order to further examine the behavior of the RNS-LRA

and randomization or RNS operations techniques and to validate the fact that certain countermeasures are necessary to prevent leakage of the implementation. The attacks explore data dependent and location dependent leakage and give similar results for all cases. When the randomized scalar or operations countermeasures are activated the results are different as expected. As an evaluation metric for the leakage of the RNS algorithm we used *Perceived Information (PI)* introduced to SCA by Renauld et al. [RSV$^+$11]. Compared to the mutual information metric, which assumes a hypothetical adversary who can perfectly profile the leakage, PI uses actual estimation procedures and practical traces to profile the Probability Density Function (PDF) of the implementation. More precisely, following the steps to estimate the PI of the implementation of RNS on the BeagleBone, as defined by Durvaux et al. in [DSVC14] we first collected profiling traces to estimate the leakage model. Then, we collected a set of test traces to estimate the PI corresponding to the actual leakage of the chip. Finally, we evaluated the estimation errors and the assumption errors from which the misclassification percentage can be calculated. The Matlab code used for counting the success rate of template attacks can be found in Appendix C.

Estimation errors occur when the number of collected traces is too low to estimate the model properly. In our approach, we avoid estimated errors by collecting 50 k traces. After alignment about 20 k traces are left, for which the template classification results were not so high. For 80% alignment threshold and the resulted 20 k aligned traces, we get success rates between $65 - 70\%$. These success rates are rather low, if we want to avoid assumption errors. Assumption errors can occur when the template model may not be able to correctly predict the distribution of samples, even after intensive profiling. If the alignment is stricter and by using a group of 10 traces for detection of the correct group instead of a single trace each time, then we can reach success rate of 99% as described later.

6.4.1 Data Dependent Leakage

Data dependent leakage is observed when the value of a secret variable can be monitored by an adversary. This happens when the variable is unprotected. Leakage can also be observed when that specific variable is protected but at the observation time the variable is sent or retrieved from a memory location in clear view. In our case, the unprotected scalar is used during scalar multiplication in the variation of the unprotected RNS implementation and the one with LRA countermeasure activated. The only case that behaves differently in data dependent leakage is the variation that uses scalar randomization or LRA randomized operations.

The key dependent assignment, the `if`-statement in Algorithm 1-step 4b (or Algorithm 2-step 4d), is the one that could be observed for this experiment. We created templates for scalars `s1 = 0x000001000···000` and `s2 = 0xFFF0···00F`. Since we wanted to observe only one instruction, we collected 50 k traces of 700

samples each. After alignment, we kept around 3 k-7 k traces (more precisely 3385 traces for the unprotected case, 3132 traces for the protected one, 7622 traces for LRA-rdm_operations), from which we used half to create templates and the other half to test if the classification in the correct template group is successful. Figure 6.25 shows the acquired traces from the protected (top) and unprotected (bottom) implementations. The selected part from each trace is used for alignment for both variations of the algorithm for 700 samples.

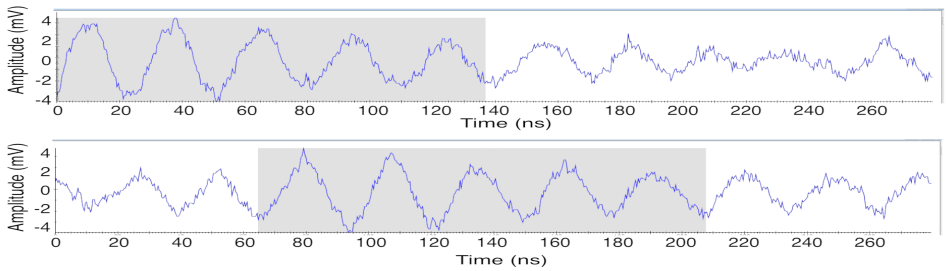

Figure (6.25) The selected part for aligning the protected and unprotected implementation

For the training of the templates and the classification we used only the aligned part for each group of traces, which is about 350 samples long as shown in Figure 6.26. The success rate of the unprotected algorithm is 91.73% for group 1 corresponding to the current bit e_i being 0, and 90.54% correctly classified traces to group 2, corresponding to the current bit being 1. The corresponding results for the LRA_rdm_point protected version is 97.47% for $e_i = 0$ and 82.12% for $e_i = 1$. The results when one countermeasure is activated give similar percentages of classification, above 90% of success rate. We did not perform the actual attack to recover each key bit, we just show here with high classification results, that with template attacks it is possible to distinguish bit 0 from bit 1.

Scalar randomization and randomized RNS operations seem to be an efficient countermeasures against data dependent template attacks. We performed the same procedure as with the previous sets of traces. More specifically, we collected 50 k traces for all variations of the implementation. Then we performed alignment, and trained templates from the aligned part of the traces. We note here that we use only the selected part to train and classify templates.

The initial results give a success rate of 65-72% for all four variations of scalar randomization, which are quite low to give us confidence for the classification results. This might be a result of the bias of the GMP randomization function. We could not achieve higher success rate for this amount of traces and it is possible that more traces would eliminate the bias and give better results. However, since we perform the attack on the blinded scalar, it is expected that the data dependent leakage

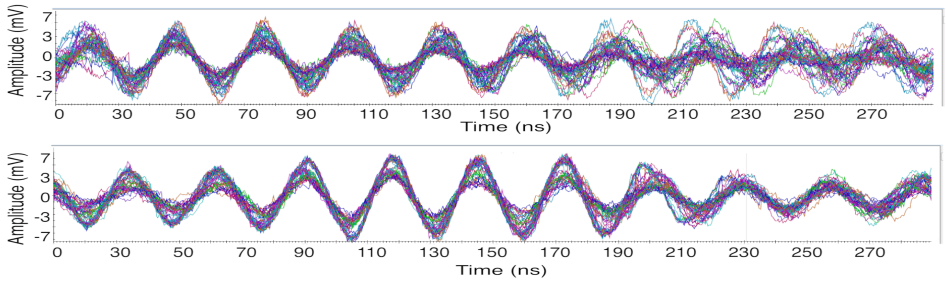

Figure (6.26) Aligned template traces for scalar assignment protected (top) and unprotected (bottom) algorithms

is very low in this case and therefore, simple classification techniques cannot separate the two sets as before. Clustering algorithms from machine learning, as applied in [OPB16, PITM14] might give better results for the scalar randomization version.

Finally, we tried to classify the template traces for the LRA_rdm_operations() function and we got $55 - 58\%$ of correct classification, which shows that by using this combination of RNS countermeasures randomizes the data enough to make template attacks hard to succeed.

6.4.2 Location Dependent Leakage

In this section, we present location dependent template attacks. The templates are created based on the storage structure that handles a key-dependent instruction, in our case the doubling during scalar multiplication. As indicated in Algorithm 28 (lines 9-14), during an MPL round an addition and a doubling of points on the curve happen every time in the same order. The only thing that differs according to the current scalar bit is the manipulated register. More specifically, when the current scalar bit is 1, then the content of storage variable R_1 is doubled, otherwise R_0 is doubled. We exploit this vulnerability and the fact that the implementation has no memory address randomization. By capturing the doubling operation, we can successfully create and classify templates for R_0 and R_1. In this way the scalar could be recovered bit-by-bit. This sort of memory access leakage is exploited in [PITM14] for the case of RSA using RNS, where the authors used unsupervised learning and clustering algorithms to classify their traces. We show here how to obtain high classification results using template attacks for the case of ECC with RNS on our software implementation.

The template classification results for all four variations of scalar multiplication vary between 87-99% and thus give a very high probability of a successfully launched template attack. We calculated the two normal distributions for $e_i = 0$ and $e_i = 1$ for every variation of the implementation and they are indeed very different

($\mathcal{N}(-24.3, 9.7)$ and $\mathcal{N}(19.6, 6.1)$).[6] This is why the misclassification percentage is very low.

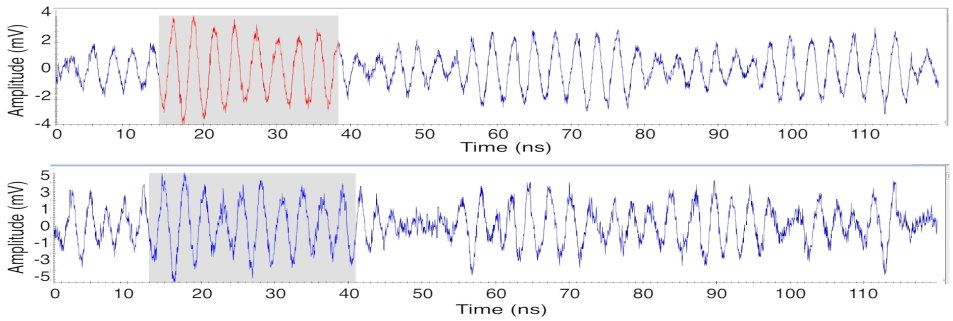

Figure (6.27) Selected area for alignment for protected and unprotected doubling

For the doubling operation, we collected 50 k traces of 3000 samples. After alignment, we used the remaining 14 k traces for the location dependent attack. We use half of the traces, that is 7 k traces of 3000 samples, for training the templates, and the remaining 7 k traces for classification.[7] Figure 6.27 shows the raw traces and selected pattern for alignment and Figure 6.28 shows the aligned traces for doubling in the case of the randomized scalar implementation. As previously, we use the 451-samples-long area to train and classify traces. The result of classification is correct with 99.44% for group 1 and 99.97% for group 2. For the case where randomization of the scalar is activated together with LRA, the situation is similar and the success rate of correct classification of templates reaches 95%. The fact that scalar randomization does not seem to be an efficient countermeasure against localized templates is also worth extra justification. We believe that the bits of the randomized scalar is what is correctly classified and can be recovered, not the initial, unblinded scalar. For the implementation `protected_rdm_scalar()` the classification results are in the range $63 - 70\%$ for various chosen alignment segments. This result is not high enough to give us the confidence of guessing correctly each key bit. Therefore, we can consider the location dependent template attack unsuccessful in this case.

When we use unified formulas, we create templates for the first addition operation (corresponding to the doubling of the value of the register). We collect 50 k traces, each one being 7 k samples-long, and we perform the template classification in Matlab as before. The results are similar to the first case with the non-unified

[6] By definition of the normal or Gaussian distribution for a random variable $\mathcal{N}(\mu, \sigma^2)$, μ indicates the mean and σ^2 the variance of the distribution.

[7] This number of traces may seem too low for an evaluation of a symmetric key implementation. For instance, Unterstein et al. in [UHSS17] performed EM location dependent templates for the case of the AES S-Box and they used 1 M traces for profiling. Acquiring 1 M is unrealistic for public-key evaluations, because it is not possible for the analysis tools to process this amount of information. Therefore, we have to draw our conclusions with the amount of traces that we can acquire.

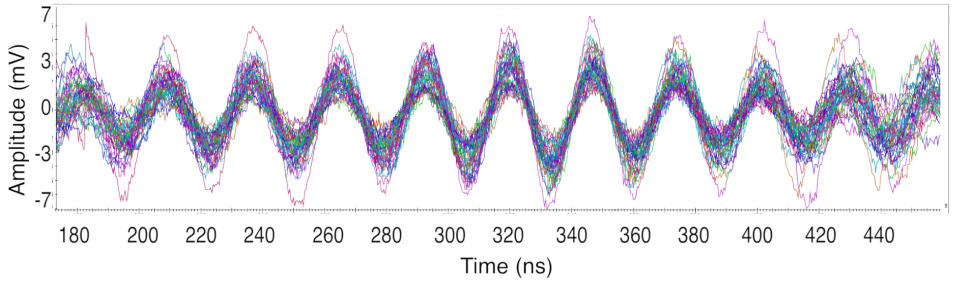

Figure (6.28) Zoom-in on aligned traces for doubling operation with randomized scalar

formulas. The classification percentages range between $72 - 91\%$ for the different implementations, but they are still high enough, in order to consider a location template attack successful.

The case of `LRA_rdm_operations` shows lower classification levels of $70-83\%$ which indicates that combining two RNS-specific countermeasures makes template attacks harder to perform. Acquiring more traces might give better results, but this might discourage the attacker, since already for 50 k traces we need 20 hours of acquisition time. In order to obtain higher classification rates, we performed stricter alignment, resulting in 8k aligned traces with segments of 370 samples. Then, we could reach $82-95\%$ classification rates, which shows that location template attacks are successful also in the presence of two RNS-specific countermeasures.

The registers mentioned in the algorithms are not really single registers that hold the result of doubling. In RNS all values are stored in chunks of 50 bits (in our case), so the result of doubling is stored in 4 different memory locations. Therefore, the location dependent leakage is not an expected result for an RNS implementation and especially with the LRA countermeasure activated. We attribute the high success rate of this attack to the leakage of the platform and the fact that those chunks of values are probably stored in capacitors next to each other.

6.5 Performance Impact of Countermeasures

In this section, we discuss the performance impact of the various countermeasures we implemented. At first, we note that performance optimizations were out of scope of our work. Therefore, we did not apply SIMD instructions or parallel computations of RNS operations. An unprotected implementation of RNS takes 0.726 sec. to perform a 192-bit scalar multiplication.

As a reference point we take the unprotected version of RNS and we measure the extra time needed as we add countermeasures. Applying point randomization costs more than performing random base permutations, because of one extra point

doubling that is necessary. So we have 1.101 sec. for point randomization instead of 1.090 for the LRA algorithm. When both countermeasures are applied, the scalar multiplication takes 1.525 sec., which is more than double compared to the unprotected version. Adding scalar randomization to all the algorithms would add a $1-8\%$ performance overhead (PO). Adding unified formulas makes our algorithms $16-30\%$ slower. Using the LRA technique with random RNS operations takes 1.278 sec. and adds 76% overhead, which is much less than the combination of LRA and randomization of the point. At the same time, this combination leaks less and preclude data dependent template attacks. It is therefore proposed as a better equivalent to traditional SCA countermeasures.

Table 6.2 shows the performance evaluation and the attack possibilities for every algorithm in terms of this work.

Table (6.2) Collective Evaluation Table. The ✗ sign shows that the corresponding algorithm does not pass the t-test or it is not protected against template attacks, while the ✓ sign shows a secure algorithm according to TVLA or against templates. The N/A indicates that this type of attack is not applicable to the algorithm. The percentage numbers show the PO compared to the unprotected RNS implementation.

Algorithm	Welch t-test r-vs-f scalar	Welch t-test r-vs-f point	TA Data	TA Location	PO
`unprotected`	✗	✗	✗	✗	0%
`rdm_point`	✗	✗	✗	✗	52%
`LRA`	✗	✗	✗	✗	50%
`protected_LRA`	✗	✓	✗	✗	110%
`unprotect_rdm_scal`	✓	N/A	✓	✗	19%
`rdm_point_rdm_scal`	✓	N/A	✓	✗	54%
`LRA_rdm_scalar`	✓	N/A	✓	✗	51%
`protected_rdm_scal`	✓	N/A	✓	✓	110%
`unprotect_unified`	✗	✗	✗	✗	19%
`rdm_point_unified`	✗	✗	✗	✗	99%
`LRA_unified`	✗	✗	✗	✗	72%
`protected_unified`	✓	✓	✗	✗	144%
`LRA_nc_rdm_operat`	✗	✓	✓	✓	76%
`LRA_nc_rdm_operat _rdm_scalar`	✓	N/A	✓	✓	76%

The TVLA experiments for RNS-MPL shows that the proposed RNS-LRA algorithm with combined countermeasures can indeed provide high levels of security compared with the simple RNS implementation or an implementation with a single countermeasure. More precisely, the systematic correlation between the power consumption and the input point can be prevented by using the classical counter-

measure of point randomization and by randomizing the base point representation that is used in every round. A very interesting observation is that using the RNS LRA representation and randomizing the input point, but not the scalar, leaks secret information. Another important remark is that the LRA countermeasure is the best trade-off between security and performance compared to traditional point blinding, which is usually $2 - 20\%$ more expensive to implement in RNS.

6.6 Conclusions

In this section we summarize the conclusions of our practical evaluation of RNS implementations with various countermeasures. Different RNS representations of elliptic curve points, randomization of the input point, scalar randomization and the regularity of MPL during scalar multiplication are good countermeasures to protect against horizontal type of attacks and SPA. However, even in the presence of these countermeasures the TVLA results show that leakage is still exploitable most of the times. When two countermeasures are combined, but the secret scalar is not randomized, there seems to be still exploitable leakage. The TVLA results are verified by high classification levels for two types of template attacks in the presence of countermeasures. When we randomize secret data, it is shown that data dependent template attacks are not successful, but the manipulated registers can still give very high classification results for the correct scalar guess.

Another important contribution of this chapter is for evaluators; they should not take the TVLA bounds as rigid, since our evaluation shows that for public-key cryptography the typical threshold of ± 4.5 is not correct. TVLA threshold should be established every time according to the distribution of the traces and the number of samples that are collected.

As future work, it would be interesting to investigate further the possibilities of location dependent template attacks in the presence of countermeasures. The classification rates for templates of the algorithms that use randomization of the RNS operations is also an interesting follow-up work that could investigate clustering and other algorithms from machine learning. Furthermore, evaluating the combination of these countermeasures in a FPGA implementation of RNS with parallel execution of the RNS operations for various moduli sizes would give further insights in the security of RNS.

Chapter 7

Conclusions

> By believing passionately in something that still does not exist, we create it. The nonexistent is whatever we have not sufficiently desired.
>
> *Nikos Kazantzakis, Report to Greco*

We conclude this thesis by summarizing the results of our research and their impact on secure ECC implementations. Future research directions inspired from this work are also provided, followed by a discussion in the end of this chapter.

7.1 Summary of Results

The main research goal of this thesis was to investigate practical and theoretically supported techniques, in order to provide side-channel resilient cryptographic implementations. We sufficiently covered this goal by proposing a new attack technique and effective countermeasures against it, by proposing new countermeasures and by evaluating an existing countermeasure with practical measurements. The main results of this research are summarized as follows:

- We presented OTA, a powerful template attack technique that is applicable to most exponentiation and scalar multiplication algorithms. The impact of the attack is verified by the fact that the German Federal Office for Information Security (BSI) includes OTA in the document "ECC-Guide Minimal Requirements for Evaluating Side-Channel Attack Resistance of Elliptic Curve Implementations" [FGI$^+$]. It is, therefore, highly advised that countermeasures against OTA should be included in a secure ECC design.
- A new approach to template matching by using classification algorithms as distinguishers is presented in Chapter 4. We showed the applicability of clas-

sification algorithms in an OTA scenario, where we successfully matched template traces to the correct target trace, in order to recover bits of the scalar.

- An important contribution of this work with direct application to security evaluations is the re-establishment of the TVLA threshold. Evaluators should not take TVLA bounds as rigid, since this threshold depends on factors such as the number of samples per trace and the sampled standard deviation of each group of traces. We showed experimentally and theoretically that new bounds should be put forward when TVLA evaluations on public key algorithms are performed. Furthermore, we provided the formulas and Matlab code to calculate this threshold, which can be applied to any public key implementation evaluation.

- Exponent splitting methods might be an effective countermeasure against certain types of side-channel attack, but their impact on performance made it prohibitive for applications with space limitations. We proposed a new exponent splitting technique, where the exponent is the XOR sum of the two shares and the cost is typically an extra register and some register copies per bit. Our method can also be applied to additive groups (ECC) and to values whose order contains long runs of bits set to 0 or 1 without any penalty on performance or security. We showed that our algorithms are secure using formal methods, MI-based evaluation and TVLA on an implementation of Boolean exponent splitting. Furthermore, we showed how an ECDSA signature can be generated by only manipulating shares of the random nonce used to secure the scheme. Our method cannot prevent leakage from intermediate values, but it can be easily combined with inexpensive solutions such as randomizing projective points, in order to provide a high level of side-channel resistance.

- RNS is already used for distributing large dynamic range computations to independent, modular operations in smaller fields. This property, based on the Chinese Remainder Theorem, is very useful for public-key cryptography, where computations in large prime fields are necessary. Apart from speeding up the computations by parallelization of operations on smaller fields, RNS has an intrinsic SCA protection mechanism, which is observed but not thoroughly studied. We performed a broad and generic evaluation of variations of RNS scalar multiplication algorithms based on the TVLA methodology and template attacks. The results of our research showed that a combination of RNS-based and traditional countermeasures is the best way to protect against side-channel leakage.

7.2 Future Research

As we discussed in Chapters 3 and 4, machine learning explores the construction of algorithms that can learn from and make predictions on data by exploiting more dimensions of the available information. Therefore, machine learning techniques can provide some other arrows in the quiver of an evaluator. Classification algorithms from machine learning are already applied to template matching with very promising results. The Least Squares Support Vector Machines (LSSVMs) is used in [HGM+11], the Bayes classifier, the k-Nearest Neighbor and SVM are used in Chapter 4 of this thesis. The results from both works showed that the data sets can be easily separated and the success rate of template attacks are much higher when classifiers from machine learning are used.

Lerman et al. [LPB+15] showed that attacks based on machine learning cannot be more efficient than template attacks in the online phase if the profiling phase is sufficiently accurate. This comes from the fact that the (mutual) information leakage estimated with a template attack exploiting a perfect model is independent of the number of "useless dimensions" if the useless leakage samples are independent of the useful ones. On the other hand, when the profiling base is relatively small, there are not enough profiling traces and the estimation of the covariance matrix cannot be computed properly. In that case, Picek et al. [PHG17] showed that evaluation using the Bayes classifier gives more accurate results than the usual profiling based on the covariance matrix.

Differential cluster analysis, introduced by Batina et al. [BGLR09], is a technique developed to detect internal collisions in the unsupervised analysis setting, and extract keys from side-channel signals. Clustering analysis and unsupervised learning was also used by Heyszl et al. [HIM+13], in order to exploit the location-based single-execution leakage of an FPGA-based implementation of an elliptic curve scalar multiplication using the k-means clustering algorithm. New attack techniques based on machine learning and a systematic evaluation of implementations based on these techniques could lead to faster leakage assessments and more efficient side-channel analysis.

In Chapter 6, we mentioned that the classification rates for templates of the algorithms that use randomization of the RNS operations is also an interesting follow-up work. It would be interesting to investigate clustering and other algorithms from machine learning in the RNS setting and further examine the possibilities of location dependent template attacks in the presence of countermeasures. Furthermore, evaluating the combination of these countermeasures in a hardware implementation of RNS with parallel execution of the RNS operations for various moduli sizes would give further insights in the security of RNS.

Finally, a very interesting project would be the research on applications of XOR exponent splitting. Further software and hardware implementations, adopting XOR splitting to real-world applications and more side-channel evaluations on different

platforms would unfold the full potential of this new and promising countermeasure.

7.3 Discussion

The readers of this thesis are convinced by now that there are various sophisticated side-channel attacks, which could threaten the security of public key implementations. Given a specific budget, in terms of space, speed, capacity, memory or financial requirements, it is impossible to protect an implementation against all possible side-channel attacks. An implementation with many countermeasures would either be too slow and impractical to use, or it would require sophisticated hardware (for instance hardware random generator) that would make it hard to fulfill in restricted environments.

On the other hand, the fact that an implementation will leak, does not mean that this leakage is always exploitable. Because of the advances in the security of implementations and the countermeasures applied on chips at the hardware level, it can be that traditional SCA techniques do not provide efficient ways to exploit a leakage. New SCA techniques inspired by the field of machine learning arise. But still, it might take months to launch a successful attack on a cryptosystem, especially when the secret keys are updated regularly or the exact details of the implementation are not known. A take-away lesson for organizations is to update secret keys and passwords on a regular basis, as this action can prevent an adversary from launching a successful attack on their systems.

The fact that secure implementations are emerging and countermeasures are applied can give us a positive feeling towards a more secure digital world. It is, however, an individual choice how aware we are on security issues and at the end, how vulnerable we can afford to be.

Appendix

A. Sage code

In this first section of the Appendix, we give the Sage scirpts that can be used to verify the correctness of the algorithm with XOR-split scalar, presented in Chapter 5.

Sagemath [OSS] is an open-source mathematical software, built out of nearly 100 open-source packages. It can be used for elementary and advanced mathematics, and it is well-suited for research. The main user language is Python. Elliptic curve computations over $\mathbb{Q}$ or over finite fields are possible in Sage, since there are many built-in functions and methods to facilitate the calculations [OSS].

The scripts are written in Python, saved with `.sage` extension, then load to the Sage server and call the relevant function. For instance

```
>> load: algorithm20.sage
>> ml_split_scalar()
```

First, we present algorithm 19, the Montgomery Ladder with XOR-split scalar, where the first bit is always set to 1. So the random register $R_{\neg b'}$ is initalized with the point $2\boldsymbol{P}$ and this is also reflected in the for-loop that starts with the second MSB.

```
#!/usr/bin sage -python

#Algorithm 19: Montgomery Ladder with XOR-split scalar on
    an Elliptic Curve
import sys

def ml_split_scalar():
  p = random_prime(2^512-1, 2^555)
  F = FiniteField(p)
  E = EllipticCurve(F,
      [F.random_element(),F.random_element()])
  P = E.random_point()
  I = E(0)
  k = randint(0,100)
  Q = k*P

  k = str(bin(k)).lstrip('0b')[::-1]
  R = [P, P, P]
  k1, a, b = []

  for i in range(0,len(k)):
    k1.append(int(k[i]))
    a.append(randint(0,1))
    b.append(k1[i] ^^ a[i])
  bp = randint(0,1)
  R[1-bp] = 2*P

#Main loop for lines 4-9 in the thesis

  for i in range(len(k1)-2,-1,-1):
    R[2] = R[1-a[i]] + R[a[i]]
    R[a[i]] = 2*R[(b[i] ^^ bp) ^^ a[i]]
    R[1-a[i]] = R[2]
    bp = b[i]

  if (R[bp]==Q):
    return 'P=', P, 'R[bp]', R[bp], 'Q', Q, 'Good!'
  else:
    return 'P=', P, 'R[bp]', R[bp], 'Q', Q, 'error'
```

Montgomery Ladder with XOR-split scalar 20, where elimination of the third register R_2 is achieved by using two auxiliary registers U_0, U_1 with precomputed values $\boldsymbol{P}$ and $-\boldsymbol{P}$.

```
#!/usr/bin sage -python

#Algorithm 20: Montgomery Ladder with XOR-split scalar II
    on an Elliptic Curve

import sys

def ml_split_scalar2():
  p = random_prime(2^512-1, 2^255)
  F = FiniteField(p)
  E = EllipticCurve(F, [F.random_element(),
      F.random_element()])
  P = E.random_point()
  M = E.random_point()
  I = E(0)
  k = randint(0,100)
  Q = k*P

  k = str(bin(k)).lstrip('0b')[::-1]
  R = [P, P]
  U = [P, -P]
  k1, a, b = []

  for i in range(0,len(k)):
    k1.append(int(k[i]))
    a.append(randint(0,1))
    b.append(k1[i] ^^ a[i])
  bp = randint(0,1)
  R[1-bp] = 2*P

#Main for loop lines 5-9 in the thesis

  for i in range(len(k1)-2,-1,-1):
    h = (bp ^^ b[i]) ^^ a[i]
    R[0] = R[bp ^^ b[i]] + R[h]
    R[1] = R[0] + U[b[i]]
    bp = b[i]
#for testing: print R[0], R[1], R[0]-R[1]

  if (R[bp] == Q):
    return 'P=', P, 'R[bp]', R[bp], 'Q', Q, 'Good!'
  else:
    return 'P=', P, 'R[bp]', R[bp], 'Q', Q, 'error'
```

This algorithm corresponds to Algorithm 21 and it removes the need for special treatment for the point at infinity $\boldsymbol{O}$ by initializing the registers with random points on the curve.

```
#!/usr/bin sage -python

#Algorithm 21: Blinded Joye's Add-Always with XOR-split
    scalar on an Elliptic Curve
import sys

def blinded_addalways():
   p = random_prime(2^512-1, 2^555)
   F = FiniteField(p)
   E = EllipticCurve(F,
       [F.random_element(),F.random_element()])
   P = E.random_point()
   M = E.random_point()
   I = E(0)
   k = randint(0,100)
   Q = k*P

   k = str(bin(k)).lstrip('0b')[::-1]
   R = [M, -M, P]

   k1 = []
   a = []
   b = []

   for i in range(0,len(k)):
      k1.append(int(k[i]))
      a.append(randint(0,1))
      b.append(k1[i] ^^ a[i])
   bp = randint(0,1)
   r = bp
   R[1-bp] = R[1-bp] + P

#Main loop for lines 3-9 in the thesis

   for i in range(0,len(k1)):
      h = (b[i] ^^ bp) ^^ a[i]
      R[2] = 2*R[1-h] + R[h]
      R[a[i]] = R[h]
      R[1-a[i]] = R[2]
      bp = b[i]

   R[r ^^ bp] = R[r ^^ bp] - M
   R[1-(r ^^ bp)] = R[1-(r ^^ bp)] + M
```

```
42    if (R[bp]==Q):
43      return 'P=', P, 'R[bp]', R[bp], 'Q', Q, 'Good!'
44    else:
45      return 'P=', P, 'R[bp]', R[bp], 'Q', Q, 'error'
```

B. Test Vector Leakage Assessment

This section includes the Matlab code used to perform TVLA for various different trace sets collected from the RNS implementationpresented in Chapter 6. Each trace set was collected with different countermeasures (or combination of countermeasures) activated, it was partitioned and processed in pieces (lines 9-18 in the code), in order to speed up the calculations. This is necessary when the traces have more than 500k samples, which is the case for public key implementations. The total number of traces corresponds to `no_pieces*no_traces`.

Each trace contains data and in the beginning there is a label `"0x0000..."` or `"0x1000..."` to indicate whether this trace has randomized input (scalar or point) or not. We alternated between random input and fixed input in a random way during acquisition. The code lines 36-44 show how the traces are split into two groups or randomized input or not. The rest of the code (lines 72-88) deals with the computation of the statistical values needed to calculate the t value for the t-test.

```
1  %modified t-test code, based on the paper by Tobias
       Scheider CHES 2015 and the Lockheed Martin formulas
       2008, to use for the public-key t-tests for RNS
2
3  clear all;
4  tic;
5  %select the t-test order
6  test_order=1;
7  max_order = 2*test_order;
8
9  %import params for fixed and random traces
10 no_traces=300; %size of the partition in traces -- change
       according to the partition choice
11 no_samples=1000000;
12 no_pieces=15;
13 offset=61; %trs file offset
14 data_size=68; %relates to the cipher\implementation
15 trs_fileID =
       fopen('protected_1msamp_trigger_doubling_Abs_align.trs')
       ;
16 traces_noB = int8(zeros( no_traces , no_samples )) ;
17 data = uint8(zeros( no_traces , 68 ));
18 status = fseek(trs_fileID, offset, 'bof');
19
20 % seperate fixed vs random traces in a group - do it for
       each collection of traces (for each piece)
21 for p=1:no_pieces
22    disp('piece number');
23    p
24    i_fixed=1;
```

```
25     i_rand=1;
26     traces_fixed = (zeros(no_traces , no_samples )) ;
27     traces_rand = (zeros(no_traces , no_samples )) ;
28     data_fixed = (zeros(no_traces , 68 )) ;
29     data_rand = (zeros(no_traces , 68 )) ;
30
31 for i = 1:no_traces,
32
33   header = uint8(fread(trs_fileID,37,'uint8')) ' ;
34   data(i,:) = uint8(fread(trs_fileID,68,'uint8')) '; %
         should be 32 for AES
35
36    if (
         (data(i,1)==hex2dec('00'))&&(data(i,2)==hex2dec('00')
         ) )
37                 traces_fixed(i_fixed,:) =
                       (fread(trs_fileID,no_samples,'uint8'));
38                 data_fixed(i_fixed,:)= data(i,:);
39                 i_fixed=i_fixed+1;
40 else
41                 traces_rand(i_rand,:) =
                       (fread(trs_fileID,no_samples,'uint8'));
42                 data_rand(i_rand,:)= data(i,:);
43                 i_rand=i_rand+1;
44    end
45
46 mid= floor(size(traces_rand,1)/2);
47 final=size(traces_rand,1);
48 traces_fixed=traces_rand(1:mid,:);
49 traces_rand=traces_rand((mid+1):final,:);
50 data_fixed=data_rand(1:mid,:);
51 data_rand=data_rand((mid+1):final,:);
52
53 end
54
55
56 if (p==1)
57     CMd_A = zeros(max_order,no_samples);
58     SMd_A = zeros(max_order,no_samples);
59     M1_A = zeros(1,no_samples);
60     CSd_A = zeros(max_order,no_samples);
61     CMd_B = zeros(max_order,no_samples);
62     SMd_B = zeros(max_order,no_samples);
63     M1_B = zeros(1,no_samples);
64     CSd_B = zeros(max_order,no_samples);
65 end
66
```

```
67
68
69  [n1(p),temp1]=size(data_fixed);
70  [n2(p),temp2]=size(data_rand);
71
72  % computation of statistical values for the t-test
73  [CMd_A, SMd_A, M1_A, CSd_A] =
        univariate_moment_computation(traces_fixed(1:n1(p),:),
74            test_order,p,n1,CMd_A, SMd_A, M1_A,CSd_A);
75  [CMd_B, SMd_B, M1_B, CSd_B] =
        univariate_moment_computation(traces_rand(1:n2(p),:),
76            test_order,p,n2,CMd_B, SMd_B, M1_B,CSd_B);
77
78
79
80  if (test_order==1)
81     mA=M1_A;
82     mB=M1_B;
83     varA=CMd_A(2,:);
84     varB=CMd_B(2,:);
85  end
86
87
88  %computing the HW t-test
89   t=t_test(mA,mB,varA,varB,sum(n1(1:p)),sum(n2(1:p)),test_order);
         %no history
90
91
92
93  end
94  close_trs(trs_fileID);
95  toc;
96
97  %ploting
98  x=1:no_samples;
99  threshold=6*ones(no_samples,1);
100 figure; plot(x,abs(t),'b');hold on;
101 plot(x,threshold,'r'); hold off;
102 xlim([0 no_samples]);
103 ylim([0 18]);
104 title('1st-order t-test');xlabel('Time
        samples');ylabel('t-statistic value');
```

C. Template Attacks on RNS software implementation

Template traces were collected by triggering either a register with value 0 or 1 (for the data leakage) or an operation of doubling/adding (for the location leakage). After importing the trace set, we split the traces into two groups. One group contains the traces that were collected when the register was manipulating the value 0 and the other group contains traces collected for the value 1. Then, we partitioned the data set into two roughly equal sets, at a 50–50 ratio. These are the training trace-set and the test trace-set. The success rate funtion `SR()` computes the success rate based on the correct classification of the test traces to one of the training sets.

```
1
2  %code based on the 'Templates Code for MiceTemplates
       Project' by Kostas Papagiannopoulos, modified and used
       to perform RNS Location and Data Template Attacks
3
4  clear all;
5  close all;
6
7  %import trace trs
8  offset=123; %trs file offset
9  data_size=68; %relates to the cipher\implementation
10 filename1 =
       '3kprotected_scalar0001_trigger_doubling_StaticAlign.trs';
11 filename2 =
       '3kprotected_scalarFF00_trigger_doubling_StaticAlign.trs';
12
13 %Number of groups used for the template building phase
14 no_groups=2;
15 no_traces = [19340 19340]; %number of traces per group
16 data1 = uint8(zeros( no_traces(1) , 68 ));
17 data2 = uint8(zeros( no_traces(2) , 68 ));
18 sdim=3000; %number of samples per trace
19 poi=1:sdim;
20
21 [S{1},status1,traces1,data1] =
       open_trs(filename1,no_traces(1),sdim,data_size,offset);
22 [S{2},status2,traces2,data2] =
       open_trs(filename2,no_traces(2),sdim,data_size,offset);
23 S{1} = traces1(:,391:758);
24 S{2} = traces2(:,391:758);
25
26 %Success rate 50-50 train/test ratio
27 partition_vec=linspace(0.5,0.5,1);
28 p_counter=1;
29 for p=partition_vec
```

```
30      for i=1:no_groups
31         t=ceil(p*size(S{i},1));
32         train_vec{i}= 1:t;
33         test_vec{i} = (t+1):size(S{i},1);
34      end
35
36      for m=1:1 %SPECIFY YOUR CHOICE
37         success_rate(p_counter,m,:)=SR(S,sdim,train_vec,
38               test_vec,m);
39         squeeze(success_rate(p_counter,m,:));
40      end
41      p_counter=p_counter+1;
42  end
43   sr=squeeze(sum(success_rate,3));
```

Bibliography

[AARR03] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM Side Channel(s). In Burton S. Kaliski, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 29–45, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cited on page 4)

[AF08] Frederic Amiel and Benoit Feix. On the BRIP Algorithms Security for RSA. In José A. Onieva, Damien Sauveron, Serge Chaumette, Dieter Gollmann, and Konstantinos Markantonakis, editors, *International Workshop on Information Security Theory and Practices: Smart Devices, Convergence and Next Generation Networks - WISTP 2008*, volume 5019 of *LNCS*, pages 136–149. Springer Berlin Heidelberg, 2008. (Cited on page 92)

[Alp10] Ethem Alpaydin. Chapter 13, Kernel Machines. In *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010. (Cited on page 41)

[aut] Automotive IoT and the Connected Car - Internet of Things Innovation. `https://internet-of-things-innovation.com/industries/automotive-iot/`. Accessed: 2019-02-02. (Cited on page 3)

[AX98] ANSI-X9.62. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). Technical report, American National Standards Institute, 1998. (Cited on pages 14 and 70)

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129, 2016. (Cited on pages 96, 97, and 115)

[BBJ+08] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted Edwards Curves. In Serge Vaudenay, editor, *Progress in Cryptology – AFRICACRYPT 2008*, volume 5023 of *LNCS*, pages 389–405. Springer, 2008. `http://cr.yp.to/papers.html#twisted`. (Cited on pages 16, 45, 135, and 147)

[BCC+12] Steve Babbage, Dario Catalano, Carlos Cid, Benne de Weger, Orr Dunkelman, Christian Gehrmann, Louis Granboulan, Tim Güneysu, Jens Hermans, Tanja Lange, Arjen Lenstra, Chris Mitchell, Mats Näslund, Phong Nguyen, Christof Paar, Kenny Paterson, Jan Pelzl, Thomas Pornin, Bart Preneel, Christian Rechberger, Vincent Rijmen, Matt Robshaw, Andy Rupp, Martin Schläffer, Serge Vaudenay, Frè Vercauteren, and Michael Ward. ECRYPT II Yearly Report on Algorithms and Keysizes. Technical report, European Network of Excellence, 09 2012. (Cited on pages 12 and 71)

[BCKL15] Daniel J. Bernstein, Chitchanok Chuengsatiansup, David Kohel, and Tanja Lange. Twisted Hessian Curves. In *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, pages 269–294, 2015. (Cited on pages 17 and 18)

[BCLN16] Joppe W. Bos, Craig Costello, Patrick Longa, and Michael Naehrig. Selecting elliptic curves for cryptography: an efficiency and security analysis. *Journal Cryptographic Engineering*, 6(4):259–286, 2016. (Cited on pages 92, 109, and 147)

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In M. Joye and J.-J. Quisquater, editors, *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004. (Cited on page 30)

[BCP+14] Lejla Batina, Lukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. Online template attacks. In *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, pages 21–36, 2014. (Cited on pages 8, 49, 57, 70, 74, 77, 79, 82, 83, 125, and 202)

[BCP+17] Lejla Batina, Łukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. Online Template Attacks. *Journal of Cryptographic Engineering*, Aug 2017. (Cited on pages 8, 49, 57, 114, and 201)

[BDE10] Jean-Claude Bajard, Sylvain Duquesne, and Milos D. Ercegovac. Combining Leak-Resistant Arithmetic for Elliptic Curves Defined over $\mathbb{F}_p$ and RNS Representation. *IACR Cryptology ePrint Archive*, 2010:311, 2010. (Cited on page 124)

[BDK97] Jean-Claude Bajard, Laurent-Stéphane Didier, and Peter Kornerup. An RNS Montgomery modular multiplication algorithm. In *Proceedings 13th IEEE Symp. on Comp. Arithmetic*, pages 234–239. IEEE Comput. Soc, 1997. (Cited on pages 124, 128, and 131)

[BDK01] Jean-Claude Bajard, Laurent-Stéphane Didier, and Peter Kornerup. Modular multiplication and base extensions in residue number systems. In *Proceedings 15th IEEE Symposium on Computer Arithmetic. ARITH-15 2001*, pages 59–65. IEEE Comput. Soc, 2001. (Cited on page 127)

[BDL01] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001. (Cited on page 4)

[BDL+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed High-security Signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012. (Cited on pages 27 and 58)

[BEG13] Jean-Claude Bajard, Julien Eynard, and Filippo Gandino. Fault Detection in RNS Montgomery Modular Multiplication. In *IEEE 21st Symp. on Comp. Arithmetic*, pages 119–126. IEEE, April 2013. (Cited on pages 124 and 128)

[BEHZ16] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan, and Vincent Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*, pages 423–442, 2016. (Cited on page 124)

[BEM16] Jean-Claude Bajard, Julien Eynard, and Nabil Merkiche. Multi-fault attack detection for RNS cryptographic architecture. In *23nd IEEE Symposium on Computer Arithmetic, ARITH 2016, Silicon Valley, CA, USA, July 10-13, 2016*, pages 16–23, 2016. (Cited on page 128)

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, volume 3958

of *LNCS*, pages 207–228. Springer Berlin Heidelberg, 2006. (Cited on pages 92 and 109)

[BGG+14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, pages 64–81, 2014. (Cited on pages 43, 119, and 138)

[BGLR09] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Differential cluster analysis. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 112–127. Springer Berlin Heidelberg, 2009. (Cited on pages 82 and 159)

[BGMW92] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David Bruce Wilson. Fast exponentiation with precomputation (extended abstract). In *EUROCRYPT*, volume 658 of *Lecture Notes in Computer Science*, pages 200–207. Springer, 1992. (Cited on page 24)

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A more efficient AES threshold implementation. In *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings*, pages 267–284, 2014. (Cited on page 43)

[BHH+15] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 368–397, 2015. (Cited on page 202)

[BILT04] Jean-Claude Bajard, Laurent Imbert, Pierre-Yvan Liardet, and Yannick Teglia. Leak Resistant Arithmetic. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES*, Lecture Notes in Computer Science, pages 62–75, 2004. (Cited on pages 124, 125, 128, 129, and 134)

[BJ02] Éric Brier and Marc Joye. Weierstraß elliptic curves and side-channel attacks. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography*, volume 2274 of *LNCS*, pages 335–345. SV, 2002. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.8.5691&rep=rep1&type=pdf. (Cited on page 120)

[BJ13] Aurélie Bauer and Éliane Jaulmes. Correlation analysis against protected SFM implementations of RSA. In G. Paul and S. Vaudenay, editors, *Progress in Cryptology - INDOCRYPT 2013 - 14th International Conference on Cryptology in India, Mumbai, India, December 7-10, 2013. Proceedings*, volume 8250 of *LNCS*, pages 98–115. Springer, 2013. (Cited on page 92)

[BJPW13] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations. In Ed Dawson, editor, *Topics in Cryptology, CT-RSA*, volume 7779 of *LNCS*, pages 1–17. Springer Berlin Heidelberg, 2013. (Cited on page 37)

[BJPW14] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal Collision Correlation Attack on Elliptic Curves. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography*, volume 8282 of *LNCS*, pages 553–570. SV, 2014. (Cited on pages 36, 46, and 92)

[BKP09] Jean-Claude Bajard, Marcelo Kaihara, and Thomas Plantard. Selected RNS Bases for Modular Multiplication. In *2009 19th IEEE Symp. on Comp. Arithmetic*, pages 25–32. IEEE, June 2009. (Cited on pages 129, 131, 132, 134, and 135)

[BL] Daniel J. Bernstein and Tanja Lange. Explicit formulas database. http://www.hyperelliptic.org/EFD/. Accessed: 2018-6-30. (Cited on pages 14, 16, 70, 76, and 77)

[BL95] Wieb Bosma and Hendrik Willem Lenstra. Complete systems of two addition laws for elliptic curves. *Journal of Number Theory*, 53(2):229 – 240, 1995. (Cited on page 14)

[BL07] Daniel J. Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In K. Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 29–50. Springer, 2007. http://cr.yp.to/papers.html#newelliptic. (Cited on page 15)

[BL09] Daniel J. Bernstein and Tanja Lange. A complete set of addition laws for incomplete Edwards curves. *IACR Cryptology ePrint Archive*, 2009:580, 2009. (Cited on page 15)

[BMP05] Jean-Claude Bajard, Nicolas Meloni, and Thomas Plantard. Efficient RNS bases for cryptography. *IMACS'05, World Congress: Scientific ComputationApplied Mathematics and Simulation,*, 07 2005. (Cited on page 129)

[Bra13] Max Bramer. Chapter 3, Introduction to Classification: Naïve Bayes and Nearest Neighbour. In *Principles of Data Mining*, Undergraduate Topics in Computer Science, pages 21–37. Springer London, 2013. (Cited on pages 40 and 41)

[BSI10] BSI. RFC 5639 - Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation. Technical report, Bundesamt für Sicherheit in der Informationstechnik (BSI), 2010. (Cited on pages 14, 71, 77, and 86)

[BT15] Karim Bigou and Arnaud Tisserand. Single base modular multiplication for efficient hardware RNS implementations of ECC. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 123–140, 2015. (Cited on page 129)

[BvdPSY14] Naomi Benger, Joop van de Pol, Nigel P. Smart, and Yuval Yarom. "ooh aah... just a little bit" : A small amount of side channel can go a long way. In *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pages 75–92, 2014. (Cited on pages 38 and 48)

[CATB18] Jerome Courtois, Lokman Abbas-Turki, and Jean-Claude Bajard. Evaluation of Resilience of randomized RNS implementation. Cryptology ePrint Archive, Report 2018/009, 2018. `https://eprint.iacr.org/2018/009`. (Cited on page 125)

[CFA+05] Henri Cohen, Gerhard Frey, Roberto Avanzi, Cristophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman and Hall CRC, 2005. (Cited on page 12)

[CFG+10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In Miguel Soriano, Sihan Qing, and Javier López, editors,

Information and Communications Security, volume 6476 of *Lecture Notes in Computer Science*, pages 46–61. Springer Berlin Heidelberg, 2010. (Cited on pages 35, 36, and 37)

[CFG+12] Christophe Clavier, Benoit Feix, Georges Gagnerot, Christophe Giraud, Mylène Roussellet, and Vincent Verneuil. ROSETTA for Single Trace Analysis. In Steven Galbraith and Mridul Nandi, editors, *Progress in Cryptology – INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 140–155. Springer Berlin Heidelberg, 2012. (Cited on pages 36 and 92)

[CGdR+15] Tom Chothia, Flavio D. Garcia, Joeri de Ruiter, Jordi van den Breekel, and Matthew Thompson. Relay Cost Bounding for Contactless EMV Payments. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, pages 189–206, 2015. (Cited on page 2)

[CHJM03] Jae Cheol Ha and Sang Jae Moon. Randomized Signed-Scalar Multiplication of ECC to Resist Power Attacks. In Burton S. Kaliski, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 551–563, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cited on page 23)

[CJ01] Christophe Clavier and Marc Joye. Universal Exponentiation Algorithm. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES*, volume 2162 of *LNCS*, pages 300–308. Springer Berlin Heidelberg, 2001. (Cited on pages 93, 94, 95, 113, and 122)

[CJ03] Mathieu Ciet and Marc Joye. (Virtually) Free Randomization Techniques for Elliptic Curve Cryptography. In S. Qing, D. Gollmann, and J. Zhou, editors, *ICICS 2003*, volume 2836 of *LNCS*, pages 348–359. Springer, 2003. (Cited on pages 93, 94, 95, and 96)

[CMCJ04] Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. Low-cost Solutions for Preventing Simple Side-channel Analysis: Side-channel Atomicity. *IEEE Transactions on Computers*, 53(6):760–768, 2004. (Cited on pages 21 and 22)

[CMO98] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Bei-*

jing, China, October 18-22, 1998, Proceedings, pages 51–65, 1998. (Cited on pages 18 and 70)

[CMV+17] Łukasz Chmielewski, Pedro Maat Costa Massolino, Jo Vliegen, Lejla Batina, and Nele Mentens. Completing the Complete ECC Formulae with Countermeasures. *Journal of Low Power Electronics and Applications*, 7(1), 2017. (Cited on pages 42, 43, 125, 137, and 138)

[CNPQ04] Mathieu Ciet, Michael Neve, Eric Peeters, and Jean-Jacques Quisquater. Parallel FPGA Implementation of RSA with Residue Number Systems - Can Side-Channel Threats be Avoided? - Extended version. Cryptology ePrint Archive, Report 2004/187, 2004. (Cited on page 124)

[Cor99] Jean-Sébastien Coron. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *LNCS*, pages 292–302. Springer Berlin Heidelberg, 1999. (Cited on pages 18, 19, 20, 32, 34, 46, 70, 74, 92, 93, 109, and 111)

[Cor10] Atmel Corporation. ATMEL AVR32UC Technical Reference Manual. ARM Doc Rev.32002F, 2010. Accessed: 2018-7-10. (Cited on page 57)

[Cor17] Jean-Sébastien Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. *IACR Cryptology ePrint Archive*, 2017:879, 2017. (Cited on page 115)

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, pages 13–28, 2002. (Cited on pages 4, 38, 48, 49, and 113)

[CS18] Gaëtan Cassiers and François-Xavier Standaert. Improved bitslice masking: from optimized non-interference to probe isolation. Cryptology ePrint Archive, Report 2018/438, 2018. `https://eprint.iacr.org/2018/438`. (Cited on page 96)

[DCE16] A. Adam Ding, Cong Chen, and Thomas Eisenbarth. Simpler, faster, and more robust t-test based leakage detection. In *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop,*

COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers, pages 163–183, 2016. (Cited on page 43)

[DGH+16] Jean-Luc Danger, Sylvain Guilley, Philippe Hoogvorst, Cédric Murdica, and David Naccache. Improving the Big Mac Attack on Elliptic Curve Cryptography. In *The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, pages 374–386, 2016. (Cited on page 36)

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976. (Cited on pages 11 and 25)

[DPN+16] Margaux Dugardin, Louiza Papachristodoulou, Zakaria Najm, Lejla Batina, Jean-Luc Danger, and Sylvain Guilley. Dismantling real-world ECC with Horizontal and Vertical Template Attacks. In *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria*, April 2016. (Cited on pages 8, 37, 49, 50, 68, 82, 83, 84, and 202)

[DS16] François Durvaux and François-Xavier Standaert. From Improved Leakage Detection to the Detection of Points of Interests in Leakage Traces. In *Advances in Cryptology - EUROCRYPT - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 240–262, 2016. (Cited on page 44)

[DSVC14] François Durvaux, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. How to Certify the Leakage of a Chip? In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 459–476, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. (Cited on page 149)

[DZD+17] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards Sound and Optimal Leakage Detection Procedure. In *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, pages 105–122, 2017. (Cited on page 138)

[Edw07] Harold M. Edwards. A Normal Form for Elliptic Curves. In Çetin K. Koç and Christof Paar, editors, *Bulletin of the American Mathematical Society*, volume 44, pages 393–422, 2007. `http://www.ams.org/journals/bull/2007-44-03/`

S0273-0979-07-01153-6/home.html. (Cited on pages 15 and 45)

[EL10] Nevine Ebeid and Rob Lambert. A New CRT-RSA Algorithm Resistant to Powerful Fault Attacks. In *Proceedings of the 5th Workshop on Embedded Systems Security*, WESS '10, pages 8:1–8:8, New York, NY, USA, 2010. ACM. (Cited on pages 100 and 122)

[Esc] Downloads Escar. Embedded security in cars. https://www.escar.info/downloads.html. Accessed: 2019-02-02. (Cited on page 3)

[ESJ+13] Mohammad Esmaeildoust, Dimitrios Schinianakis, Hamid Javashi, Thanos Stouraitis, and Keivan Navi. Efficient RNS Implementation of Elliptic Curve Point Multiplication Over GF(p). *IEEE Trans. VLSI Syst.*, 21(8):1545–1549, 2013. (Cited on pages 129, 132, 134, and 135)

[FGI+] Dirk Feldhusen, Max Gebhardt, Georg Illies, Michael Kasper, Manfred Lochter, Richard Petri, Oliver Stein, Wolfgang Thumser, and Guntram Wicke. ECC-guide: Minimal Requirements for Evaluating Side-Channel attack resistance of Elliptic Curve Implementations. (Cited on page 157)

[FHY07] Jianqing Fan, Peter Hall, and Qiwei Yao. To How Many Simultaneous Hypothesis Tests Can Normal, Student's t or Bootstrap Calibration Be Applied? *Journal of the American Statistical Association*, 102:1282–1288, 2007. (Cited on page 44)

[FIP01] FIPS. *Security Requirements for Cryptographic Modules*. PUB-NIST, 2001. Annex A: Approved Security Functions (19 May 2005); Annex B: Approved Protection Profiles (04 November 2004); Annex C: Approved Random Number Generators (31 January 2005); Annex D: Approved Key Establishment Techniques (30 June 2005). Supersedes FIPS PUB 140-1, 1994 January 11. (Cited on page 26)

[FJ10] Reza Rezaeian Farashahi and Marc Joye. Efficient Arithmetic on Hessian Curves. In *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, pages 243–260, 2010. (Cited on page 17)

[FKSK15] Apostolos P. Fournaris, Nicolaos Klaoudatos, Nicolas Sklavos, and Christos Koulamas. Fault and Power Analysis Attack Resistant RNS based Edwards Curve Point Multiplication. In *Proceedings of the*

2nd Workshop on Cryptography and Security in Computing Systems, CS2 at HiPEAC 2015, Amsterdam, Netherlands, January 19-21, 2015, pages 43–46, 2015. (Cited on pages 124 and 127)

[Fou17] Apostolos P Fournaris. *Fault and Power Analysis Attack Protection Techniques for Standardized Public Key Cryptosystems*, pages 93–105. Springer International Publishing, Cham, 2017. (Cited on pages 128, 134, and 135)

[Fou18a] OpenSSL Software Foundation. OpenSSL Cryptography and SSL TLS Toolkit. `https://www.openssl.org/docs/`, 2018. Accessed: 2018-4-4. (Cited on page 26)

[Fou18b] Apostolos P. Fournaris. RNS_LRA_EC_scalar Multiplier, 2018. `https://github.com/afournaris/RNS_LRA_EC_Scalar_Multiplier`. (Cited on pages 9, 133, 136, and 140)

[FPBS16] Apostolos P. Fournaris, Louiza Papachristodoulou, Lejla Batina, and Nicolaos Sklavos. Residue Number System as a side channel and fault injection attack countermeasure in elliptic curve cryptography. In *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–4, April 2016. (Cited on pages 9, 124, 125, 126, 128, 133, and 202)

[FPS17] Apostolos P. Fournaris, Louiza Papachristodoulou, and Nicolaos Sklavos. Secure and Efficient RNS Software Implementation for Elliptic Curve Cryptography. In *2017 IEEE Eur. Symp. Secur. Priv. Work.*, pages 86–93. IEEE, April 2017. (Cited on pages 9, 125, 126, 130, 133, 135, and 201)

[FRV14] Benoit Feix, Mylène Roussellet, and Alexandre Venelli. Side-channel Analysis on Blinded Regular Scalar Multiplications. In W. Meier and D. Mukhopadhyay:, editors, *INDOCRYPT 2014*, volume 8885 of *LNCS*, pages 3–20. Springer, 2014. (Cited on pages 37, 92, and 96)

[FV03] Pierre-Alain Fouque and Frédéric Valette. The doubling attack — Why upwards is better than downwards. In Colin D. Walter, Çetin K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *LNCS*, pages 269–280. SV, 2003. (Cited on pages 35, 36, 46, 47, and 51)

[FV06] Guillaume Fumaroli and David Vigilant. Blinded Fault Resistant Exponentiation. In *Fault Diagnosis and Tolerance in Cryptography,*

Third International Workshop - FDTC, pages 62–70, 2006. (Cited on page 134)

[FV12a] Junfeng Fan and Ingrid Verbauwhede. An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost. In David Naccache, editor, *Cryptography and Security: From Theory to Applications*, volume 6805 of *LNCS*, pages 265–282. SV, 2012. (Cited on page 124)

[FV12b] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://eprint.iacr.org/2012/144`. (Cited on page 124)

[Gal12] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012. (Cited on page 12)

[Gir] Damien Giry. Bluekrypt - Cryptographic Key Length Recommendation. (Cited on page 124)

[Gir06] Christophe Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Trans. on Computers*, 55(9):1116–1120, 2006. (Cited on page 134)

[GJJR11] Gilbert Goodwill, Benjamin Jun, Joshua Jaffe, and Pankaj Rohatgi. A Testing Methodology for Side-channel Resistance Validation, 2011. NIST Non-Invasive Attack Testing Workshop. (Cited on pages 42, 43, 94, 119, 120, 125, and 137)

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *LNCS*, pages 251–261. Springer, 2001. (Cited on pages 4 and 30)

[Gor98] Daniel M. Gordon. A Survey of Fast Exponentiation Methods. *J. Algorithms*, 27(1):129–146, 1998. (Cited on page 22)

[Gou01] Louis Goubin. A Sound Method for Switching between Boolean and Arithmetic Masking. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems CHES*, volume 2162 of *LNCS*, pages 3–15. Springer, 2001. (Cited on page 111)

[Gui10] Nicolas Guillermin. A High Speed Coprocessor for Elliptic Curve Scalar Multiplications over $\mathbb{F}_p$. *Cryptographic Hardware and Embedded Systems - CHES*, pages 48–64, 2010. (Cited on page 130)

[Gui11] Nicolas Guillermin. A Coprocessor for Secure and High Speed Modular Arithmetic. *IACR Cryptology ePrint Archive*, 2011. (Cited on pages 124 and 130)

[Ham15] Michael Hamburg. Ed448-Goldilocks, a New Elliptic Curve. Cryptology ePrint Archive, Report 2015/625, 2015. `http://eprint.iacr.org/2015/625`. (Cited on pages 92 and 109)

[HGM+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine Learning in Side-channel Analysis: A First Study. *Journal of Cryptographic Engineering*, 1(4):293–302, 2011. (Cited on page 159)

[HGS01] Nick A. Howgrave-Graham and Nigel P. Smart. Lattice Attacks on Digital Signature Schemes. *Design, Codes and Cryptography*, 23:283–290, 2001. (Cited on page 108)

[HIM+13] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, and Georg Sigl. Clustering Algorithms for Non-profiled Single-Execution Attacks on Exponentiations. In *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, pages 79–93, 2013. (Cited on pages 39, 47, 83, and 159)

[HKT15] Neil Hanley, HeeSeok Kim, and Michael Tunstall. Exploiting Collisions in Addition Chain-based Exponentiation Algorithms Using a Single Trace. In K. Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 431–448. Springer, 2015. (Cited on pages 46, 92, and 95)

[HMA+08] Naofumi Homma, Atsushi Miyamoto, Takafumi Aoki, Akashi Satoh, and Adi Shamir. Collision-based Power Analysis of Modular Exponentiation Using Chosen-message Pairs. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES*, volume 5154 of *LNCS*, pages 15–29. Springer Verlag, 2008. (Cited on page 46)

[HMH+12] Johann Heyszl, Stefan Mangard, Benedikt Heinz, Frederic Stumpf, and Georg Sigl. Localized Electromagnetic Analysis of Cryptographic Implementations. In *Topics in Cryptology - CT-RSA*, pages 231–244, 2012. (Cited on page 126)

[HMV03] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003. (Cited on pages 12 and 24)

[Hoe15] Jochen Hoenicke. Extracting the Private Key from a Trezor. `https://jochen-hoenicke.de/crypto/trezor-power-analysis/`, 2015. Accessed: 2019-01-08. (Cited on page 3)

[HPS18] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An Improved RNS Variant of the BFV Homomorphic Encryption Scheme. Cryptology ePrint Archive, Report 2018/117, 2018. `https://eprint.iacr.org/2018/117`. (Cited on page 124)

[HS13] Michael Hutter and Peter Schwabe. NaCl on 8-bit AVR microcontrollers. In Amr Youssef and Abderrahmane Nitaj, editors, *Progress in Cryptology – AFRICACRYPT 2013*, volume 7918 of *LNCS*, pages 156–172. Springer, 2013. (Cited on pages 57, 58, and 64)

[HTM11] Neil Hanley, Michael Tunstall, and William P. Marnane. Using Templates to Distinguish Multiplications from Squaring Operations. *International Journal of Information Security*, 10(4):255–266, 2011. (Cited on page 83)

[HWCD08] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted Edwards Curves Revisited. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 326–343. SV, 2008. (Cited on pages xxviiixxviii, 15, 16, 58, and 66)

[HWCD09] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Faster Group Operations on Elliptic Curves. In *Proceedings of the Seventh Australasian Conference on Information Security - Volume 98*, AISC '09, pages 7–20, Darlinghurst, Australia, Australia, 2009. Australian Computer Society, Inc. (Cited on page 17)

[IISO10] Masami Izumi, Jun Ikegami, Kazuo Sakiyama, and Kazuo Ohta. Improved Countermeasures Against Address-Bit DPA for ECC Scalar Multiplication. In G. De Michell, B. M. Al-Hashimi, W. Müller, and E. Macii, editors, *DATE 2010*, pages 981–984. IEEE, 2010. (Cited on pages 93, 99, 102, 113, and 117)

[IIT02] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenada. Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. In Burton S. Kaliski Jr., Çetin K. Koç, and Christof

Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 129–143. Springer, 2002. (Cited on page 93)

[IIT03] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenada. A Practical Countermeasure Against Address-Bit Differential Power Analysis. In Colin D. Walter, Çetin K. Koç, and Christof Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 382–396. Springer, 2003. (Cited on pages 93, 100, 113, and 120)

[IMT] Tetsuya Izu, Bodo Möller, and Tsuyoshi Takagi. Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks. In *Progress in Cryptology - INDOCRYPT 2002, Third International Conference on Cryptology in India, Hyderabad, India, December 16-18, 2002*. (Cited on page 17)

[Int] Mordor Intelligence. Hardware Wallet Market - Segmented by Type (USB, NFC, and Bluetooth) and Geography - Growth, Trends, and Forecast (2019 - 2024). Accessed: 2019-01-08. (Cited on page 3)

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 463–481, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cited on page 96)

[JL12] Marc Joye and Tancrède Lepoint. Partial Key Exposure on RSA with Private Exponents Larger Than N. In *Information Security Practice and Experience - 8th International Conference, ISPEC 2012, Hangzhou, China, April 9-12, 2012. Proceedings*, pages 369–380, 2012. (Cited on page 122)

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). `https://web.archive.org/web/20170921160141/http://cs.ucsb.edu/~koc/ccs130h/notes/ecdsa-cert.pdf`, 2001. Accessed: 2018-19-10, Refer to section 6.2 for ECDSAPUBKEY, and section 7 for ECDSASIGN and ECDSARECOVER. (Cited on page 26)

[Joy03] Marc Joye. *Elliptic Curves and Side-Channel Analysis*, volume 4, pages 283–306. ST Journal of Systems Research, 2003. (Cited on pages 70, 74, and 88)

[Joy07] Marc Joye. Highly Regular Right-to-Left Algorithms for Scalar Multiplication. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th In-*

ternational Workshop, Vienna, Austria, September 10-13, 2007, Proceedings, volume 4727 of *LNCS*, pages 135–147. SV, 2007. (Cited on pages 20, 50, 55, 103, and 107)

[JQ01] Marc Joye and Jean-Jacques Quisquater. Hessian Elliptic Curves and Side-Channel Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, number Generators, pages 402–410, 2001. (Cited on page 17)

[JRW11] Joshua Jaffe, Pankaj Rohatgi, and Marc Witteman. Efficient Side Channel Testing for Public Key Algorithms: RSA Case Study, 2011. NIST Non-Invasive Attack Testing Workshop. (Cited on pages 43, 125, and 142)

[JT01a] Marc Joye and Christophe Tymen. Compact Encoding of Non-adjacent Forms with Applications to Elliptic Curve Cryptography. In *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, pages 353–364, 2001. (Cited on page 22)

[JT01b] Marc Joye and Christophe Tymen. Protections Against Differential Analysis for Elliptic Curve Cryptography. In *Cryptographic Hardware and Embedded Systems - CHES, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, pages 377–390, 2001. (Cited on page 89)

[JWAG18] Shalabh Jain, Qian Wang, Md Tanvir Arafin, and Jorge Guajardo. Probing Attacks on Physical Layer Key Agreement for Automotive Controller Area Networks (Extended Version). *CoRR*, abs/1810.07305, 2018. (Cited on page 3)

[JY02] Marc Joye and Sung Ming Yen. The Montgomery Powering Ladder. In Burton S. Kaliski, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *LNCS*, pages 291–302. SV, 2002. (Cited on pages 21, 88, 97, and 133)

[Kel98] Thomas Kelly. The Myth Of The Skytale. *Cryptologia*, 22(3):244–260, 1998. (Cited on page 1)

[KGG+18] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael

Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. *meltdownattack.com*, 2018. (Cited on page 3)

[KGP+09] Peter Karsmakers, Benedikt Gierlichs, Kristiaan Pelckmans, Katrien De Cock, Johan Suykens, Bart Preneel, and Bart De Moor. Side Channel Attacks on Cryptographic Devices as a Classification Problem. Technical report, KU Leuven, 2009. (Cited on page 83)

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999. (Cited on pages 4, 5, 30, 32, 46, and 95)

[KKSS00] Shinichi Kawamura, Masanobu Koike, Fumihiko Sano, and Atsushi Shimbo. Cox-Rower Architecture for Fast Parallel Montgomery Multiplication. In *Advances in Cryptology— EUROCRYPT*, 2000. (Cited on pages 127 and 128)

[KKYH10] HeeSeok Kim, Tae Hyun Kim, Joong Chul Yoon, and Seokhie Hong. Practical Second-Order Correlation Power Analysis on the Message Blinding method and its Novel Countermeasure for RSA. *ETRI Journal*, 32(1):102–111, 2010. (Cited on pages 92, 95, and 99)

[Kob87] Neal Koblitz. *Elliptic Curve Cryptosystems*, volume 48, pages 203–209. Mathematics of Computation, 1987. (Cited on pages 12 and 26)

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996. (Cited on page 30)

[LD99] Julio López and Ricardo Dahab. Improved Algorithms for Elliptic Curve Arithmetic in GF(2n). In Stafford Tavares and Henk Meijer, editors, *Selected Areas in Cryptography*, pages 201–212. Springer Berlin Heidelberg, 1999. (Cited on page 24)

[LL94] Chae Hoon Lim and Pil Joong Lee. More Flexible Exponentiation with Precomputation. In Y. Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *LNCS*, pages 95–107. Springer, 1994. (Cited on page 24)

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovasz. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261:515–534, 1982. (Cited on page 46)

[LMSS14] Manfred Lochter, Johannes Merkle, Jörn-Marc Schmidt, and Torsten Shültze. Requirements for Standard Elliptic Curves. `http://www.ecc-brainpool.org/`, 2014. Accessed: 2018-7-10. (Cited on pages 68 and 92)

[LMV+13] Liran Lerman, Stephane Fernandes Medeiros, Nikita Veshchikov, Cédric Meuter, Gianluca Bontempi, and Olivier Markowitch. Semi-Supervised Template Attack. In *Constructive Side-Channel Analysis and Secure Design - 4th International Workshop, COSADE 2013, Paris, France, March 6-8, 2013, Revised Selected Papers*, pages 184–199, 2013. (Cited on page 83)

[LPB+15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis). In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design (COSADE)*, Lecture Notes in Computer Science (LNCS), pages 20–33. Springer, 4 2015. (Cited on pages 39 and 159)

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *meltdownattack.com*, 2018. (Cited on page 3)

[LTT15] Duc-Phong Le, Chik How Tan, and Michael Tunstall. Randomizing the Montgomery Powering Ladder. In Raja Naeem Akram and Sushil Jajodia, editors, *Information Security Theory and Practice - 9th IFIP WG 11.2 International Conference WISTP 2015*, volume 9311 of *LNCS*, pages 169–184. Springer, 2015. (Cited on page 101)

[mbe] ARM mbed. mbedTLS v2.2.0. `https://tls.mbed.org/`. Accessed: 2018-7-10. (Cited on pages 24 and 70)

[MBO+05] Elke De Mulder, Pieter Buysschaert, Sıddıka Berna Örs, Peter Delmotte, Bart Preneel, Guy Vandenbosch, and Ingrid Verbauwhede. Electromagnetic Analysis Attack on an FPGA Implementation of an Elliptic Curve Cryptosystem. In *IEEE International Conference on Computer as a tool*, pages 1879–1882, November 2005. Belgrade, Serbia & Montenegro. DOI: 10.1109/EURCON.2005.1630348, `http://www.sps.ele.tue.nl/members/m.j.bastiaans/spc/demulder.pdf`. (Cited on page 38)

[MCW+18] Alyssa Milburn, Santiago Cordoba, Nils Wiersma, Ramiro Pareja, and Niek Timmers. There will be Glitches: Extracting and Anayzing Automotive Firmware Efficienlty. In *Black Hat USA 2018, Las Vegas, NV, USA*, 2018. (Cited on page 3)

[MDAB10] Steven J. Murdoch, Saar Drimer, Ross J. Anderson, and Mike Bond. Chip and PIN is Broken. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berkeley/Oakland, California, USA*, pages 433–446, 2010. (Cited on page 2)

[MDS99] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power Analysis Attacks of Modular Exponentiation in Smartcards. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 1999. (Cited on page 93)

[MHMP13] Elke De Mulder, Michael Hutter, Mark E. Marson, and Peter Pearson. Using Bleichenbacher's Solution to the Hidden Number Problem to Attack Nonce Leaks in 384-Bit ECDSA. In Guido Bertoni and Jean-Sébastian Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES*, volume 8086 of *LNCS*, pages 435–452. SV, 2013. (Cited on pages 38 and 46)

[Mil85] Victor S. Miller. Use of Elliptic Curves in Cryptography. In *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, pages 417–426, 1985. (Cited on page 12)

[MMS01] David May, Henk L. Muller, and Nigel P. Smart. Random Register Renaming to Foil DPA. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware for Embedded Systems - CHES*, volume 2162 of *LNCS*, pages 28–38. Springer, 2001. (Cited on page 93)

[MO90] François Morain and Jorge Olivos. Speeding Up the Computations on an Elliptic Curve Using Addition-Subtraction Chains. *Informatique Théorique et Applications*, 24:531–544, 1990. (Cited on page 22)

[MO09] Marcel Medwed and Elisabeth Oswald. Template attacks on ECDSA. In Kyo-Il Chung, Kiwook Sohn, and Moti Yung, editors, *Information Security Applications*, volume 5379 of *LNCS*, pages 14–27. Springer, 2009. (Cited on pages 4, 38, 46, 47, and 51)

[MOBW13] Luke Mather, Elisabeth Oswald, Joe Bandenburg, and Marcin Wójcik. Does My Device Leak Information? An a priori Statistical Power Analysis of Leakage Detection Tests. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, pages 486–505, 2013. (Cited on pages 43 and 44)

[Mon87] Peter L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177):243–264, 1987. (Cited on pages 16 and 21)

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer-Verlag, Berlin, Heidelberg, 2007. (Cited on pages 31, 33, 34, 51, and 99)

[MT18] Alyssa Milburn and Niek Timmers. Efficient Reverse Engineering of Automotive Firmware. In *ESCAR USA conference 2018, Ypsilanti, MI, USA*, 2018. (Cited on page 3)

[Nak09] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. `http://bitcoin.org/bitcoin.pdf`, 2009. Accessed: 2018-7-10. (Cited on page 26)

[Nat09] National Institute of Standards and Technology (NIST). Recommended Elliptic Curves for Federal Government Use. In the appendix of FIPS 186-3, June 2009. Accessed: 2018-7-10. (Cited on pages 68, 70, 92, 108, and 109)

[Ned18] Betaalvereniging Nederland. Fraude in betalingsverkeer, 2018. Accessed 2018-10-10. (Cited on page 2)

[New18] Lily Hay Newman. The Elite INTEL Team still Fighting Meltdown and Spectre. `https://www.wired.com/story/intel-meltdown-spectre-storm/`, 2018. Accessed: 2019-01-08. (Cited on page 3)

[NIS13] NIST. FIPS publication 186-4 - Digital Signature Standard (DSS). Technical report, National Institute of Standards and Technology (NIST), July 2013. (Cited on pages 14, 71, 77, and 109)

[NLD15] Erick Nascimento, Julio López, and Ricardo Dahab. Efficient and secure elliptic curve cryptography for 8-bit avr microcontrollers. In Rajat Subhra Chakraborty, Peter Schwabe, and Jon Solworth, editors, *Security, Privacy, and Applied Cryptography Engineering: 5th*

International Conference, SPACE 2015, Jaipur, India, October 3-7, 2015, Proceedings, pages 289–309, Cham, 2015. Springer International Publishing. (Cited on pages 42, 43, 125, 137, and 138)

[NNTW05] David Naccache, Phong Q. Nguyen, Michael Tunstall, and Claire Whelan. Experimenting with Faults, Lattices and the DSA. In S. Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 16–28. Springer, 2005. (Cited on page 108)

[NP16] Christophe Negre and Thomas Plantard. Efficient Regular Modular Exponentiation Using Multiplicative Half-Size Splitting. *Journal of Cryptographic Engineering*, pages 1–9, 2016. (Cited on page 93)

[NS03] Phong Q. Nguyen and Igor E. Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Designs, Codes and Cryptography*, 30(2):201–217, Sep" 2003. (Cited on page 38)

[NSS04] David Naccache, Nigel P. Smart, and Jacques Stern. Projective Coordinates Leak. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 257–267. Springer, 2004. (Cited on page 58)

[OKS00] Katsuyuki Okeya, Hiroyuki Kurumatani, and Kouichi Sakurai. Elliptic Curves with the Montgomery-Form and Their Cryptographic Applications. In *Public Key Cryptography, Third International Workshop on Practice and Theory in Public Key Cryptography, PKC 2000, Melbourne, Victoria, Australia, January 18-20, 2000, Proceedings*, pages 238–257, 2000. (Cited on page 17)

[OM07] Elisabeth Oswald and Stefan Mangard. Template Attacks on Masking - Resistance Is Futile. In Masayuki Abe, editor, *Topics in Cryptology - CT-RSA*, volume 4377 of *Lecture Notes in Computer Science*, pages 243–256. Springer, 2007. (Cited on page 113)

[ÖOP03] Siddika Berna Örs, Elisabeth Oswald, and Bart Preneel. Power-Analysis Attacks on an FPGA - First Experimental Results. In *Cryptographic Hardware and Embedded Systems - CHES, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, pages 35–50, 2003. (Cited on pages xxviixxvii and 32)

[OPB16] Elif Özgen, Louiza Papachristodoulou, and Lejla Batina. Classification Algorithms for Template Matching. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016,*

McLean, VA, USA, 2016. (Cited on pages 8, 39, 49, 50, 53, 82, 151, and 202)

[OS02] Katsuyuki Okeya and Kouichi Sakurai. A Second-Order DPA Attack Breaks a Window-Method Based Countermeasure against Side Channel Attacks. In A. H. Chan and V. Gligor, editors, *ISC 202*, volume 2433 of *LNCS*, pages 389–401. Springer, 2002. (Cited on page 92)

[OSS] Various Developers Open Source Software. Sagemath Mathematical Software. `http://www.sagemath.org/index.html`. Accessed: 2018-10-11. (Cited on page 161)

[PBM17] Louiza Papachristodoulou, Lejla Batina, and Nele Mentens. Recent Developments in Side-Channel Analysis on Elliptic Curve Cryptography Implementations. In Nicolas Sklavos, Ricardo Chaves, Giorgio Di Natale, and Francesco Regazzoni, editors, *Hardware Security and Trust: Design and Deployment of Integrated Circuits in a Threatened Environment*. Springer International Publishing, Incorporated, 2017. (Cited on pages 8 and 201)

[PFPB19] Louiza Papachristodoulou, Apostolos P. Fournaris, Kostas Papagianopoulos, and Lejla Batina. Practical Evaluation of Protected Residue Number System Scalar Multiplication (to appear). In *Cryptographic Hardware and Embedded Systems – CHES*, TCHES, 2019. (Cited on pages 9, 126, and 201)

[PHG17] Stjepan Picek, Annelie Heuser, and Sylvain Guilley. Template Attack Versus Bayes Classifier. *Journal of Cryptographic Engineering*, 7(4):343–351, 2017. (Cited on page 159)

[PITM13] Guilherme Perin, Laurent Imbert, Lionel Torres, and Phillipe Maurine. Electromagnetic Analysis on RSA Algorithm Based on RNS. In *2013 Euromicro Conference on Digital System Design (DSD)*, number 1, pages 345–352, 2013. (Cited on pages 125 and 143)

[PITM14] Guilherme Perin, Laurent Imbert, Lionel Torres, and Philippe Maurine. Attacking Randomized Exponentiations Using Unsupervised Learning. In *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, pages 144–160, 2014. (Cited on pages 39, 82, and 151)

[PP95] Karl C. Posch and Reinhard Posch. Modulo Reduction in Residue Number Systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(5):449–454, May 1995. (Cited on page 124)

[Pro18] Horizon2020 Reassure Project. Understanding Leakage Detection - Free Tutorial co-organized with CARDIS 2018 . `http://reassure.eu/leakage-detection-tutorial/`, 2018. Accessed: 2019-01-20. (Cited on page 44)

[PV04] Dan Page and Frederik Vercauteren. Fault and Side-Channel Attacks on Pairing Based Cryptography. Cryptology ePrint Archive, Report 2004/283, 2004. `https://eprint.iacr.org/2004/283`. (Cited on page 34)

[PV17] Kostas Papagiannopoulos and Nikita Veshchikov. Mind the Gap: Towards Secure 1st-Order Masking in Software. In *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, pages 282–297, 2017. (Cited on page 43)

[QS01] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In Isabelle Attali and Thomas Jensen, editors, *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security*, volume 2140 of *E-SMART '01*, pages 200–210, London, UK, UK, 2001. Springer-Verlag. (Cited on pages 4 and 30)

[RCB16] Joost Renes, Craig Costello, and Lejla Batina. Complete addition formulas for prime order elliptic curves. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 403–428, 2016. (Cited on pages 14 and 45)

[Rei60] George W. Reitwiesner. Binary arithmetic. volume 1 of *Advances in Computers*, pages 231 – 308. Elsevier, 1960. (Cited on page 22)

[Res00] Certicom Research. Standards for efficient cryptography, SEC 1: Elliptic curve cryptography, September 2000. Version 1.0. (Cited on page 14)

[Risa] Riscure. Inspector SCA Tool. `https://www.riscure.com/security-tools/inspector-sca/`. Accessed: 2018-6-14. (Cited on pages 57, 74, and 140)

[Risb] Riscure. Power Tracer. https://www.riscure.com/product/power-tracer/. Accessed: 2018-6-14. (Cited on page 57)

[Riv91] Ronald L. Rivest. Cryptography and Machine Learning. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances In Cryptology – ASIACRYPT*, volume 739 of *Lecture Notes in Computer Science*, pages 427–439. Springer, 1991. (Cited on page 82)

[Riv11] Matthieu Rivain. Fast and Regular Algorithms for Scalar Multiplication over Elliptic Curves. *IACR Cryptology ePrint Archive*, 2011:338, 2011. (Cited on pages 18, 72, and 88)

[RM] Research and Markets. Global IoT in Automotive Market (2018-2023). https://www.researchandmarkets.com/research/fk7fbf/iot_in_the_global?w=4. Accessed: 2019-02-02. (Cited on page 3)

[RO04] Christian Rechberger and Maria Elisabeth Oswald. Practical Template Attacks. In Chae Hoon Lim and Moti Yung, editors, *Information Security Applications*, volume 3325 of *Lecture Notes in Computer Science*, pages 440 – 456. Springer, 2004. (Cited on page 38)

[RS01] Tanja Römer and Jean-Pierre Seifert. Information Leakage Attacks Against Smart Card Implementations of the Elliptic Curve Digital Signature Algorithm. In Isabelle Attali and Thomas Jensen, editors, *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security*, volume 2140 of *LNCS*, pages 211–219. SV, 2001. (Cited on pages 38, 46, and 48)

[RSA78] Ron Rivest, Adi Shamir, and Leonard M. Adleman. Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. (Cited on pages 11 and 30)

[RSV+11] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pages 109–128, 2011. (Cited on page 149)

[SBB+18] Niels Samwel, Lejla Batina, Guido Bertoni, Joan Daemen, and Ruggero Susella. Breaking Ed25519 in WolfSSL. In *Topics in Cryptology - CT-RSA*, pages 1–20, 2018. (Cited on page 34)

[Sch14] Werner Schindler. Exclusive exponent blinding may not suffice to prevent timing attacks on RSA. Cryptology ePrint Archive, Report 2014/869, 2014. `http://eprint.iacr.org/`. (Cited on page 122)

[SHKS15] Robert Specht, Johann Heyszl, Martin Kleinsteuber, and Georg Sigl. Improving Non-profiled Attacks on Exponentiations Based on Clustering and Extracting Leakage from Multi-channel High-Resolution EM Measurements. In *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, pages 3–19, 2015. (Cited on page 39)

[SI11] Werner Schindler and Kouichi Itoh. Exponent Blinding Does Not Always Lift (Partial) SPA Resistance to Higher-Level Security. In J. Lopez and G. Tsudik, editors, *ACNS 2011*, volume 6715 of *LNCS*, pages 73–90. Springer, 2011. (Cited on pages 95 and 122)

[Sil94] Joseph H. Silverman. *Advanced Topics in the Arithmetic of Elliptic Curves*. 151. Springer-Verlag, New York, 1994. (Cited on page 12)

[Sil09] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. 106. Springer-Verlag, New York, 2009. (Cited on pages 12 and 13)

[SM16] Tobias Schneider and Amir Moradi. Leakage assessment methodology - extended version. *J. Cryptographic Engineering*, 6(2):85–99, 2016. (Cited on page 43)

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 443–461, 2009. (Cited on pages 94 and 115)

[SOP08] Nigel P. Smart, Elisabeth Oswald, and Daniel Page. Randomised Representations. *IET Proceedings on Information Security*, 2(2):19–27, 2008. (Cited on pages 92 and 96)

[SP] The Statistical Portal. PayTV subscribers in the USA. Accessed: 2018-10-10. (Cited on page 3)

[SS13] Dimitrios Schinianakis and Thanos Stouraitis. Hardware Fault Attack Handling in RNS-based Montgomery Multipliers. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 3042–3045. IEEE, 2013. (Cited on page 124)

[ST67] Nicholas S. Szabo and Richard I. Tanaka. *Residue Arithmetic and Its Applications to Computer Technology*. McGraw-Hill Book Company, New York, 1967. (Cited on page 128)

[Sta03] Martijn Stam. On Montgomery-Like Representations for Elliptic Curves over GF(2k). In *Public Key Cryptography — PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings", editor="Desmedt, Yvo G.*, pages 240–254. Springer Berlin Heidelberg, 2003. (Cited on pages 16 and 21)

[Ste09] William Stein. *Elementary Number Theory: Primes, Congruences, and Secrets*. Springer-Verlag, New York, 1 edition, 2009. (Cited on page 13)

[SWP03] Kai Schramm, Thomas Wollinger, and Christof Paar. A New Class of Collision Attacks and Its Application to DES. In Thomas Johansson, editor, *Fast Software Encryption*, volume 2887 of *LNCS*, pages 206–222. SV, 2003. (Cited on page 46)

[TG16] Michael Tunstall and Gilbert Goodwill. Applying TVLA to Public Key Cryptographic Algorithms. *IACR Cryptology ePrint Archive*, 2016. `https://eprint.iacr.org/2016/513`. (Cited on pages 42, 43, 125, and 137)

[TK06] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Third Edition*. Academic Press, Inc., Orlando, FL, USA, 2006. (Cited on page 83)

[TPP18] Michael Tunstall, Louiza Papachristodoulou, and Kostas Papagiannopoulos. Boolean Exponent Splitting. *IACR Cryptology ePrint Archive*, 2018:1226, 2018. (Cited on pages 9 and 201)

[UHSS17] Florian Unterstein, Johann Heyszl, Fabrizio De Santis, and Robert Specht. Dissecting Leakage Resilient PRFs with Multivariate Localized EM Attacks - A Practical Security Evaluation on FPGA. In *COSADE*, volume 10348 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2017. (Cited on page 152)

[vdBOYPdR] Jordi van den Breekel, Diego Ortiz-Yepes, Erik Poll, and Joeri de Ruiter. EMV in a Nutshell - Technical Report. Technical report. Acessed: 2018-10-10. (Cited on page 2)

[Wal01] Colin D. Walter. Sliding Windows Succumbs to Big Mac Attack. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *LNCS*, pages 286–299. Springer, 2001. (Cited on pages 34, 35, 46, and 92)

[Wit17] Witteman Marc. When Hardware Attacks Scale, 2017. (Cited on page 3)

[WKK13] Erich Wenger, Thomas Korak, and Mario Kirschbaum. Analyzing Side-Channel Leakage of RFID-Suitable Lightweight ECC Hardware. In Michael Hütter and Jörn-Marc Schmidt, editors, *Radio Frequency Identification*, volume 8262 of *LNCS*, pages 128–144. SV, 2013. (Cited on page 46)

[WMPW98] Erik De Win, Serge Mister, Bart Preneel, and Michael J. Wiener. On the Performance of Signature Schemes Based on Elliptic Curves. In J.-P. Buhler, editor, *ANTS 1998*, volume 1423 of *LNCS*, pages 252–266. Springer, 1998. (Cited on pages 92, 100, 107, and 122)

[Won15] David Wong. Timing and Lattice Attacks on a Remote ECDSA OpenSSL Server: How Practical Are They Really? Cryptology ePrint Archive, Report 2015/839, 2015. `https://eprint.iacr.org/2015/839`. (Cited on page 46)

[WS14] Andreas Wiemers Werner Schindler. Power Attacks in the Presence of Exponent Blinding. *Journal of Cryptographic Engineering*, 4(4):213–236, 2014. (Cited on page 122)

[WvWM11] Marc F. Witteman, Jasper G.J. van Woudenberg, and Federico Menarini. Defeating RSA Multiply-Always and Message Blinding Countermeasures. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA*, volume 6558 of *LNCS*, pages 77–88. Springer, 2011. (Cited on pages 34, 92, and 95)

[Xil] Xilinx. Xilinx Zynq-7000 SoC ZC702 Evaluation Kit. (Cited on pages 119 and 120)

[YKMH05] Sung-Ming Yen, Lee-Chun Ko, Sang Jae Moon, and Jae Cheol Ha. Relative Doubling Attack Against Montgomery Ladder. In Dong Ho

Won and Seungjoo Kim, editors, *Information Security and Cryptology – ICISC 2005*, volume 3935 of *LNCS*, pages 117–128. Springer, 2005. (Cited on pages 35 and 46)

[ZDD+17] Liwei Zhang, A. Adam Ding, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards Sound and Optimal Leakage Detection Procedure. *IACR Cryptology ePrint Archive*, 2017:287, 2017. (Cited on pages 42, 43, 44, and 137)

[ZWMZ14] Zhenbin Zhang, Liji Wu, Zhaoli Mu, and Xiangmin Zhang. A Novel Template Attack on wNAF Algorithm of ECC. In *2014 Tenth International Conference on Computational Intelligence and Security*, pages 671–675, Nov 2014. (Cited on page 47)

Index

List of Publications

Book chapter

1. Louiza Papachristodoulou, Lejla Batina, Nele Mentens, *Recent Developments in Side-Channel Analysis on Elliptic Curve Cryptography Implementations*, Chapter 3 in book "Hardware Security and Trust: Design and Deployment of Integrated Circuits in a Threatened Environment" by editors N. Sklavos, R. Chaves, G. Di Natale, F. Regazzoni, Springer International Publishing, Incorporated, 2017. [PBM17]

International Journals

1. Lejla Batina, Lukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, Michael Tunstall, *Online Template Attacks*, Journal of Cryptographic Engineering, 2017 [BCP+17]

Conference Proceedings & Preprints

1. Louiza Papachristodoulou, Apostolos P. Fournaris, Kostas Papagiannopoulos, Lejla Batina, *Practical Evaluation of Protected Residue Number System Scalar Multiplication*, IACR Transactions on Cryptographic Hardware and Embedded Systems, Volume 2019, Issue 1 (accepted-to appear in 2019) [PFPB19]

2. Michael Tunstall, Louiza Papachristodoulou, Kostas Papagiannopoulos, *Boolean Exponent Splitting*, IACR Cryptology ePrint Archive 2018:1226, 2018 [TPP18]

3. Apostolos P. Fournaris, Louiza Papachristodoulou, Nicolas Sklavos, *Secure and Efficient RNS Software Implementation for Elliptic Curve Cryptography*, IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2017, pp. 86-93, 2017 [FPS17]

4. Apostolos P. Fournaris, Louiza Papachristodoulou, Lejla Batina, Nicolas Sklavos, *Residue Number System as a Side-Channel and Fault Injection Attack Countermeasure in Elliptic Curve Cryptography*, International Conference

on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2016 [FPBS16]

5. Elif Özgen, Louiza Papachristodoulou, Lejla Batina, *Template Attacks Using Classification Algorithms*, 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 242-247, 2016 [OPB16]

6. Margaux Dugardin, Louiza Papachristodoulou, Zakaria Najm, Lejla Batina, Jean-Luc Danger, Sylvain Guilley, *Dismantling Real-World ECC with Horizontal and Vertical Template Attacks*, COSADE 2016, LNCS 9689, 2016 [DPN+16]

7. Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, Zooko Wilcox O'Hearn, *SPHINCS: Practical Stateless Hash-Based Signatures*, Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, LNCS 9056, pp. 368-397, 2015 [BHH+15]

8. Lejla Batina, Lukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, Michael Tunstall, *Online Template Attacks*, Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, pp.21-34, 2014 [BCP+14]

Curriculum Vitae

Louiza Papachristodoulou was born on 19^{th} July 1984 in Athens, Greece. In 2002 she graduated from the Lyceum of Agios Stefanos and she entered the National Technical University of Athens after successfully completing the Panhellenic exams. She specialized in Mathematics and Computer Science, doing her thesis in the area of topology and set theory, more precisely on *Martin's Axiom*. In the summer semester of 2016 she studied in the Technical University of Berlin, Germany, with a scholarship from the Erasmus Inter-university Cooperation Program. She obtained the Diploma (equivalent to Masters degree) in Applied Mathematical and Physical Sciences in 2008.

In 2008, Louiza was awarded a talent full-scholarship for the 2-years MSc Program in Information Security Technology (IST) by the Technical University of Eindhoven. IST was a special master track of the Kerckhoff's Institute in cooperation with Radboud University Nijmegen and University of Twente. After completing the study program in all three universities, she did her final project in *Secure Multiparty Computations on AES* in Philips Research under the supervision of dr.ir. Berry Schoenmakers.

From 2010 until 2014, Louiza worked as a Security Architect and Cryptographer in Compumatica Secure Networks in Uden, The Netherlands. Designing systems with balance between security and usability, writing proof-of-concept proposals and developing the security architecture for products that require governmental certification were some of the main tasks assigned to her in this job.

In 2013, she started her PhD research in the Digital Security Group of Radboud University Nijmegen, under the supervision of Professor Lejla Batina. The first year of her PhD was done part-time, combined with her work at Compumatica. Since July 2014, she focused full-time on her PhD project on *Side-channel attacks and countermeasures on Elliptic Curve Cryptography*, funded by the STW-project SIDES. During her PhD, she had the chance to visit research laboratories and universities for collaboration. She visited Télécom ParisTech twice (November 2014 and April 2015) to work with the group of Jean-Luc Danger funded by the ICT COST Action TRUDEVICE IC1204. In terms of TRUDEVICE, she also did a Short Term Scientific Mission in the University of Patras, Greece, working with the group of Nicolas Sklavos. In 2015, she was an Engineering Intern at Ram-

bus Security-Cryptography Research, San Francisco, USA, where she worked on electromagnetic analysis of mobile devices using asymmetric algorithms. In 2016, Louiza was awarded with the Christine Mohrmann scholarship from Radboud University, in order to investigate the security and side-channel resistance of Bitcoin in real-world applications. She used this scholarship to perform research with the Security Group of Aggelos Kiayias at the University of Edinburgh, UK (February-April 2017) and the Computer Science Group of Yuval Yarom at the University of Adelaide, Australia (February-March 2018). Since February 2019 she works as a Senior Research Analyst in Cyber Security at NavInfo Europe B.V., providing security solutions for the automotive industry.